Zebra variations and ChatGPT

Tanuj Sood

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

BA (Mod) in Computer Science

Supervisor: Tim Fernando

April 2025

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Tanuj Sood

April 18, 2025

Zebra variations and ChatGPT

Tanuj Sood, BA (Mod) in Computer Science
University of Dublin, Trinity College, 2025

Supervisor: Tim Fernando

ABSTRACT

The research investigates the hypothesis proposed by Mahowald et al. (2024), asserting that large language models (LLMs) possess superior formal linguistic competence compared to functional linguistic competence. To rigorously test this assertion, the study employs Zebra Puzzles—a category of constraint satisfaction problems—as a benchmark. The methodology involves two distinct approaches: directly prompting an LLM to solve the puzzles in natural language and instructing the LLM to convert puzzle clues into structured JSON inputs compatible with the Z3 constraint solver. The performance of the LLM-generated solutions and conversions is assessed against established ground truths, utilizing metrics such as accuracy, consistency, and correctness of generated constraints.

Further, the study explores advanced prompting techniques, including Chain of Thought (CoT) and Multi-Shot prompting, to enhance LLM performance. The findings demonstrate that LLM's consistently performs better at converting natural language puzzle descriptions into structured constraint inputs than solving the puzzles directly. This outcome aligns with prior research and underscores the dissociation between linguistic generation capabilities and functional reasoning proficiency in current LLMs. The study contributes insights into the limitations and potential avenues for enhancing reasoning capabilities in LLMs, emphasizing the utility of structured prompting techniques and external computational verification tools.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Tim Fernando, for his invaluable guidance, insightful feedback, and continuous support throughout the course of this project. His expertise and encouragement have been instrumental in shaping this work.

Tanuj Sood

University of Dublin, Trinity College
April 2025

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The recent advent and rapid advancement of Large Language Models (LLMs), such as OpenAI's GPT series, have significantly transformed the field of natural language processing. These models have demonstrated exceptional proficiency in various linguistic tasks, particularly those related to grammar, syntax, and semantic structuring—collectively referred to as formal linguistic competence Mahowald et al. (2024). However, despite impressive performance in these areas, the functional linguistic competence of these models—covering logical reasoning, structured comprehension, and practical problem-solving—remains comparatively inconsistent and inadequately understood Mahowald et al. (2024); Giadikiaroglou et al. (2024).

Recent advancements suggest potential improvements in LLM performance for tasks involving commonsense reasoning, and arithmetic computations Brown et al. (2020); Chowdhery et al. (2022); Bubeck et al. (2023). Nevertheless, structured logical reasoning tasks, which demand structured interpretation, and adherence to logical constraints, continue to pose significant challenges. Previous attempts at directly solving structured logical puzzles, specifically Zebra puzzles also known as Einstein's riddles or logic grid puzzles, using LLMs have reported modest correctness rates—around 8.33%, Groza (2023) and correctness rates of approximately 23% for baseline newer models highlighting persistent limitations in the functional reasoning capabilities of current LLMs Berman et al. (2024).

An alternative, more linguistically focused approach to evaluating LLM competence involves leveraging these models to convert puzzle clues expressed in natural language into structured formal inputs for symbolic solvers. Constraint solvers, such as Z3 developed by Microsoft Research Z3TheoremProver (2012), offer reliable, efficient theorem-proving capabilities based on Satisfiability Modulo Theories (SMT)De Moura and Bjørner (2008). The accuracy of an LLM's translation from natural language descriptions to solver-compatible representations thus serves as a decent measure of its formal linguistic competence.

Prompt engineering techniques, like Chain-of-Thought (CoT) prompting and Multi-Shot prompting, have shown potential for enhancing both direct reasoning and puzzle-conversion capabilities by explicitly guiding models to articulate intermediate reasoning steps Wei et al. (2022). These techniques can potentially bridge the gap between formal and functional linguistic competencies by explicitly structuring reasoning processes within the models.

In this thesis, Zebra puzzles serve as a structured benchmark to test the hypothesis posed by Mahowald et al. Mahowald et al. (2024), that LLMs exhibit superior linguistic competence compared to functional competence. By systematically evaluating both the accuracy of LLM's in translating puzzle descriptions into structured solver inputs, and its capability to directly solve puzzles through natural language reasoning, this research aims to provide a clearer understanding of the strengths and limitations of contemporary LLMs in handling structured logical reasoning tasks.

1.1 Research Goals

The primary goal of this research is to empirically evaluate the hypothesis proposed by Mahowald et al. Mahowald et al. (2024), which claims that Large Language Models (LLMs) demonstrate greater proficiency in formal linguistic tasks—such as translating natural language into structured, solver-compatible representations—compared to functional linguistic tasks involving direct logical reasoning. Further, this thesis investigates whether this claim remains valid following the emergence of advanced reasoning-focused models, such as OpenAI's o-series models and DeepSeek's Reasoner model, which have explicitly aimed at improving logical reasoning capabilities.

The research aims to answer two primary questions:

1. Does the assertion that LLMs perform better on linguistic tasks than on functional reasoning tasks hold true when assessed through structured logical constraint problems, specifically Zebra puzzles?

2. Can advanced prompt engineering techniques—specifically, Chain-of-Thought (CoT) and multi-shot prompting—effectively enhance the accuracy of these state-of-the-art

models in both linguistic and functional reasoning contexts?

3. Contribute a representative snapshot of current LLM reasoning capabilities on structured logical tasks, providing a reference point for future comparison as reasoning-oriented models continue to improve and evolve.

Through systematic experimentation and comparative analysis of direct puzzle-solving abilities and the accuracy of translating natural language puzzle constraints into structured JSON formats compatible with constraint solvers (such as Z3), this research seeks to provide a clear understanding of contemporary LLMs' strengths and limitations in structured reasoning and linguistic tasks. While Zebra puzzles may not encompass the full breadth of functional reasoning challenges faced in naturalistic settings, their well-defined logical structure and constrained format make them an adequate and practical benchmark for evaluating the comparative capabilities of LLMs in linguistic versus functional competence. By evaluating and contrasting these competencies within this controlled context, the study contributes valuable insights toward understanding and advancing the abilities of next-generation language models.

1.2 Overview of the Report

This dissertation is structured into five main chapters. Chapter 2 (*Related Work*) provides a comprehensive overview of relevant literature, encompassing the foundational concepts of LLMs, constraint satisfaction problems, reasoning evaluations, and various prompt engineering strategies. Chapter 3 (*Design and Implementation*) outlines the methodologies employed in designing and validating puzzle instances, describes the experimental setups for testing linguistic and functional competencies, and details the integration and implementation of the Z3 constraint solver. Chapter 4 (*Evaluation*) systematically analyses the experimental outcomes, focusing on metrics such as solution accuracy, the correctness of constraint translations, and performance improvements attributed to prompt engineering techniques. Chapter 5 (*Conclusion*) synthesizes key findings, discusses implications for future LLM research, and outlines prospective directions aimed at refining and enhancing the functional reasoning capabilities of large language models.

By clearly delineating the comparative effectiveness of direct reasoning versus constraint model translation in LLMs, this thesis aims to contribute valuable insights into the functional linguistic competence of contemporary language models, ultimately informing strategies for advancing AI reasoning capabilities toward greater practical utility.

Chapter 2

State of the Art

This chapter surveys the key concepts, technologies, and research developments that form the foundation for this dissertation. It aims to contextualize the work by highlighting relevant advances in language models, constraint solving, and prompt engineering—fields that intersect in this research to evaluate linguistic and functional competence in Large Language Models (LLMs).

The chapter begins with a background section that introduces five central components: the evolution of LLMs and recent advances aimed at improving reasoning capabilities; the theoretical distinction between linguistic and functional competence; the nature of Constraint Satisfaction Problems (CSPs) with a focus on Zebra puzzles as a structured testbed; the Z3 constraint solver as a formal verification tool; and finally, prompt engineering techniques such as Chain-of-Thought and multi-shot prompting, which are explored in this study as potential means to enhance reasoning performance.

The second major section of the chapter reviews closely-related work across these areas. The discussion is organized thematically, drawing comparisons between how different studies approach similar challenges—such as improving LLM reasoning, integrating LLMs with symbolic systems, and leveraging prompt design strategies.

Finally, the chapter concludes with a comparative summary that highlights recurring patterns, differentiating assumptions, and limitations in the current body of work. This sets the stage for the experimental investigation that follows in later chapters by clarifying how this dissertation contributes to and expands upon the existing state of the art.

2.1 Background

2.1.1 Large Language Models and Their Recent Evolutions

Large Language Models (LLMs) represent a major breakthrough in natural language processing (NLP), enabling machines to understand and generate human language with remarkable fluency and accuracy. These models are built upon the transformer architecture Vaswani et al. (2017), and are trained on massive text corpora using self-supervised learning objectives such as next-token prediction. Over successive generations, LLMs have demonstrated increasingly sophisticated capabilities across a wide range of linguistic tasks, from machine translation and summarization to code generation and multi-turn dialogue.

Early transformer-based models, such as GPT-2 and GPT-3, showcased impressive few-shot and in-context learning abilities Brown et al. (2020). These models were capable of generalizing to unseen tasks with minimal examples, sparking significant interest in their potential for general-purpose reasoning. More recent models, including GPT-4 and GPT-o-series OpenAI (2025), have introduced substantial architectural and performance improvements. These include longer context windows, enhanced instruction-following abilities, and multimodal integration OpenAI (2023).

Alongside OpenAI's developments, other research organizations have introduced reasoning-focused models aimed at overcoming some of the reasoning limitations observed in earlier generations. Models such as Claude by Anthropic (2025), DeepSeek-Reasoner DeepSeek-AI et al. (2025), and PaLM Chowdhery et al. (2022) exemplify ongoing efforts to build LLMs capable of more structured, coherent, and logically consistent reasoning. These models often incorporate reinforcement learning from human feedback (RLHF) Ouyang et al. (2022), fine-tuning with specialized benchmarks, and architectural refinements to improve multi-step reasoning and factual consistency.

While recent advances have demonstrated promise in tasks requiring commonsense reasoning and general knowledge retrieval Brown et al. (2020); Chowdhery et al. (2022); Bubeck et al. (2023), the capabilities of LLMs in handling complex deductive problems remain uncertain. This limitation in our understanding is especially critical, as systematic reasoning underpins many real-world applications. Performance improvements on benchmarks do not necessarily translate to genuine deductive capabilities, particularly in tasks that involve symbolic logic, constraint manipulation, or multi-step inference.

Indeed, evaluating the actual reasoning ability of LLMs remains a complex challenge. Scaling alone has proven insufficient for solving tasks requiring symbolic manipulation or multi-hop reasoning Rae et al. (2021). LLMs are often prone to superficial pattern matching and hallucinations, which can be masked by their fluency in language generation. As Rae et al. Rae et al. (2021) and Xu et al. Xu et al. (2023) argue, traditional NLP benchmarks often fail to expose these deeper reasoning deficiencies. These limitations become particularly visible in tasks with explicit logical structures, such as Zebra puzzles, where consistent constraint tracking is required.

This dissertation focuses on evaluating such reasoning capabilities by applying LLMs to structured logic problems—specifically, Zebra puzzles. These puzzles require models to interpret and manage multiple interacting constraints and serve as a focused testbed for comparing linguistic competence (e.g., translating natural language clues into formal representations) against functional reasoning ability (e.g., directly solving the puzzle). The central research question is whether the recent advancements in LLM design and training yield measurable improvements in functional competence, or if performance remains dominated by linguistic pattern-matching.

2.1.2 Linguistic Competence vs. Functional Competence

A central theme of this dissertation is the dissociation between two facets of language model performance: *linguistic competence* and *functional competence*. This theoretical framework is drawn from cognitive science, particularly the work of Mahowald et al. Mahowald et al. (2024), who argue that LLMs demonstrate strong formal linguistic abilities—such as grammatical accuracy, syntactic coherence, and contextually appropriate word use—while often failing to exhibit consistent performance in reasoning, problem-solving, and other functionally grounded tasks.

Linguistic competence refers to a model's ability to generate language that adheres to rules of grammar, syntax, and vocabulary usage. It is concerned with form rather than meaning, and LLMs have consistently excelled in this domain. Their training on massive text corpora allows them to produce fluent and coherent outputs, often indistinguishable from human language in form. This linguistic prowess underpins their success in tasks like summarization, translation, and question-answering—domains that reward surface-level fluency.

In contrast, **functional competence** encompasses deeper reasoning abilities, including logical inference, numerical calculation, structured decision-making, and commonsense reasoning. These abilities are critical for tasks involving symbolic or deductive processing, such as solving math problems or logic puzzles. As Mahowald et al. note, high linguistic fluency can create the illusion of genuine understanding, even when the model fails at coherent reasoning.

To better understand how this dissociation manifests in practice, we can examine recent empirical results that quantify LLM performance across both competence types. On the linguistic side, models consistently perform well on tasks that test syntactic and formatting adherence. For instance, LLMs have achieved near-human accuracy on syntactic benchmarks such as BLiMP and SyntaxGym, which probe knowledge of grammatical structures through minimal pair evaluations and targeted syntactic phenomena, respectively Warstadt et al. (2020); Gauthier et al. (2020). Similarly, in the IFEval benchmark—which evaluates a model's ability to follow explicit formatting instructions like "include keyword x" or "use format y"—state-of-the-art models have scored close to 90% under strict evaluation criteria Zhou et al. (2023). This benchmark isolates formal linguistic competence by focusing on surface-level instruction following rather than content generation, and its results are echoed by evaluations on the Open LLM Leaderboard, where leading models routinely score at or near this threshold Fourrier et al. (2024).

In contrast, performance on functionally demanding reasoning tasks remains markedly lower. For example, Groza Groza (2023) reports that GPT-3.5 achieved only an 8.33% correctness rate when solving Zebra puzzles directly—tasks that demand structured logical inference across multiple constraints. Even more recent evaluations by Berman et al. Berman et al. (2024), using baseline non-reasoning models such as GPT-4, achieved only around 23% correctness. These results underscore the persistent difficulty LLMs face when required to perform multi-step reasoning or manage symbolic constraints, reinforcing the distinction between linguistic fluency and functional problem-solving ability.

This project evaluates both dimensions of competence using a controlled experimental setup. By comparing how LLMs perform in direct puzzle-solving versus how well they translate puzzle constraints into a formal JSON format for Z3, this research seeks to quantitatively assess the extent to which the dissociation proposed by Mahowald et al. holds true in the context of structured logical reasoning tasks.

2.1.3 Constraint Satisfaction Problems and Zebra Puzzles

Constraint Satisfaction Problems (CSPs) are a class of computational problems where the goal is to find values for a set of variables that satisfy a collection of constraints. Formally, a CSP consists of:

- A finite set of variables $\{X_1, X_2, ..., X_n\}$,
- A domain of possible values for each variable,
- A set of constraints specifying allowable combinations of values.

CSPs are ubiquitous in artificial intelligence and operations research, forming the foundation for applications such as scheduling, planning, resource allocation, and combinatorial optimization. Solving a CSP involves searching through the space of possible assignments to find those that satisfy all constraints, often leveraging techniques like backtracking, constraint propagation, and heuristics Prosser (1993).

One specific and well-known instance of a CSP is the **Zebra puzzle** (also referred to as "Einstein's riddle" or a logic grid puzzle). Zebra puzzles typically involve determining the attributes of several entities based on a series of interdependent clues. For example, the puzzle may describe five adjacent houses, each painted a different color and inhabited by people of different nationalities with distinct pets, drinks, and cigarette brands. The goal is to deduce who owns the zebra, or who drinks water, based solely on the given constraints.

The following is a classic version of the Zebra puzzle, which originally appeared in *Life International* in 1962 Wikipedia contributors (2025):

1.There are five houses.
2.The Englishman lives in the red house.
3.The Spaniard owns the dog.
4.Coffee is drunk in the green house.
5.The Ukrainian drinks tea.
6.The green house is immediately to the right of the ivory house.
7.The Old Gold smoker owns snails.
8.Kools are smoked in the yellow house.
9.Milk is drunk in the middle house.
10.The Norwegian lives in the first house.
11.The man who smokes Chesterfields lives in the house next to the man with the fox.

12.Kools are smoked in the house next to the house where the horse is kept.
13.The Lucky Strike smoker drinks orange juice.
14.The Japanese smokes Parliaments.
15.The Norwegian lives next to the blue house.
Now, who drinks water? Who owns the zebra?

House	1	2	3	4	5
Color	Yellow	Blue	Red	Ivory	Green
Nationality	Norwegian	Ukrainian	Englishman	Spaniard	Japanese
Drink	**Water**	Tea	Milk	Orange juice	Coffee
Smoke	Kools	Chesterfield	Old Gold	Lucky Strike	Parliament
Pet	Fox	Horse	Snails	Dog	**Zebra**

Table 2.1: Solution to the puzzle

Zebra puzzles are particularly attractive for AI reasoning research because they possess several useful properties:

- They are *closed-world*, finite-domain problems with deterministic outcomes.
- They require multi-hop, cross-variable logical reasoning.
- Their constraints are usually presented in natural language, enabling tests of both language understanding and symbolic reasoning.

Unlike arithmetic or commonsense questions that may admit multiple interpretations or rely on world knowledge, Zebra puzzles have clearly defined rules and a single correct answer—making them an ideal benchmark for evaluating structured reasoning capabilities in LLMs.

Previous approaches to solving Zebra puzzles in AI have often relied on manual encoding of rules into formal languages such as Prolog or SMT-based constraint solvers Prosser (1993). More recent neuro-symbolic systems, such as those by Berman et al. Berman et al. (2024), use language models in conjunction with constraint-guided solvers to bridge natural language understanding and formal reasoning. However, even with strong language modeling components, these systems continue to struggle with ambiguous phrasing, incomplete reasoning, or generating internally inconsistent intermediate steps.

In this dissertation, Zebra puzzles are employed as a structured and interpretable testbed for measuring both linguistic and functional competence in LLMs. Because they are expressed in natural language yet grounded in a well-defined logical framework with a unique solution, they offer a controlled setting for evaluating the dissociation between a model's ability to convert linguistic descriptions into formal representations versus its ability to reason through and solve the underlying problem. However, their utility as an evaluative tool comes with limitations. Zebra puzzles are highly constrained, domain-specific, and syntactically simple, which may not fully reflect the complexity and ambiguity of natural language in real-world tasks. Moreover, their deterministic nature limits the evaluation of probabilistic or open-ended reasoning, and their reliance on clearly formulated clues may not capture the challenges posed by vagueness, underspecification, or commonsense inference. As such, while they offer a clean and interpretable environment for controlled testing, their findings should be interpreted with care when extrapolating to broader reasoning contexts beyond structured logic puzzles.

2.1.4 The Z3 Constraint Solver

Z3 is a high-performance Satisfiability Modulo Theories (SMT) solver developed by Microsoft Research, designed to determine the satisfiability of logical formulas under various background theories, such as arithmetic, arrays, and bit-vectors De Moura and Bjørner (2008). It has become one of the most widely adopted tools in formal verification, program analysis, and automated theorem proving due to its speed, flexibility, and expressive power. Z3 is capable of solving complex logical constraints encoded in formats such as SMT-LIB or custom APIs, and it supports both satisfiability checking and model generation.

In the context of this dissertation, Z3 is used not to assess the functional reasoning abilities of LLMs directly, but rather to serve as a formal verification backend that evaluates the correctness of structured inputs generated by LLMs. Specifically, LLMs are prompted to translate natural language clues from Zebra puzzles into structured JSON representations programmatically compiled into Z3-compatible constraints. Z3 subsequently verifies whether these constraints are consistent and whether the resulting solution matches the ground truth.

This setup provides a rigorous and objective means of evaluating linguistic competence in models. Because Z3 operates over a symbolic and formally defined logic space, even minor deviations in the LLM's parsing or interpretation of clues—such as misassigning a variable, overlooking a negation, or failing to capture a dependency—can result in unsatisfiable constraint sets or incorrect solutions. Thus, the task of generating Z3-ready constraint models from natural language serves as a precise and unforgiving test of a model's ability to extract and formalize rule-based information.

Moreover, using Z3 as a downstream verifier eliminates ambiguities common in evaluating natural language generation tasks. The model's output is not judged on fluency or style, but solely on whether it encodes the intended logic correctly. In this way, Z3 allows for the automation of linguistic evaluation under symbolic constraints—turning a qualitative task into a verifiable, binary one.

In summary, Z3 is not used here as a reasoning engine for the language model, but rather as an impartial adjudicator for validating the linguistic-to-logical mapping generated by the LLM. This design choice enables a focused evaluation of formal linguistic competence by isolating the translation step from the reasoning process itself.

2.1.5 Prompt Engineering Techniques

In many real-world applications, language models are used in a black-box setting, where their internal weights and training processes are inaccessible, and the only method of interaction is through their input-output interface. In such contexts, the way a prompt is written can significantly influence the quality, accuracy, and structure of the model's response. Prompt engineering refers to the practice of strategically crafting these inputs to better align the model's output with the user's intent, especially in tasks that require structured reasoning or adherence to format-specific instructions.

In recent years, various prompting strategies have emerged to enhance the reasoning capabilities of LLMs. Among them, *Chain-of-Thought* (CoT) prompting and *multi-shot prompting* have demonstrated significant promise for improving model performance on tasks requiring multi-step inference, commonsense reasoning, and symbolic manipulation. These techniques serve as indirect ways of expanding the model's functional competence by modifying how the task is framed rather than changing the model itself.

The following subsections describe these two techniques in detail, providing both theoretical context and empirical justification for their use in this dissertation.

Chain-of-Thought Prompting

Chain-of-Thought (CoT) prompting is a method that encourages models to generate intermediate reasoning steps before arriving at a final answer similar to what a human might do. Rather than responding with a single direct output, the model is prompted to "think aloud," reasoning through the problem step-by-step. This technique was introduced by Wei et al. Wei et al. (2022) and has since gained widespread adoption across reasoning-intensive benchmarks. It builds on the insight that LLMs often possess latent reasoning ability that can be activated by structurally prompting the model to articulate its process DataCamp (2024).

A basic example involving a math word problem:

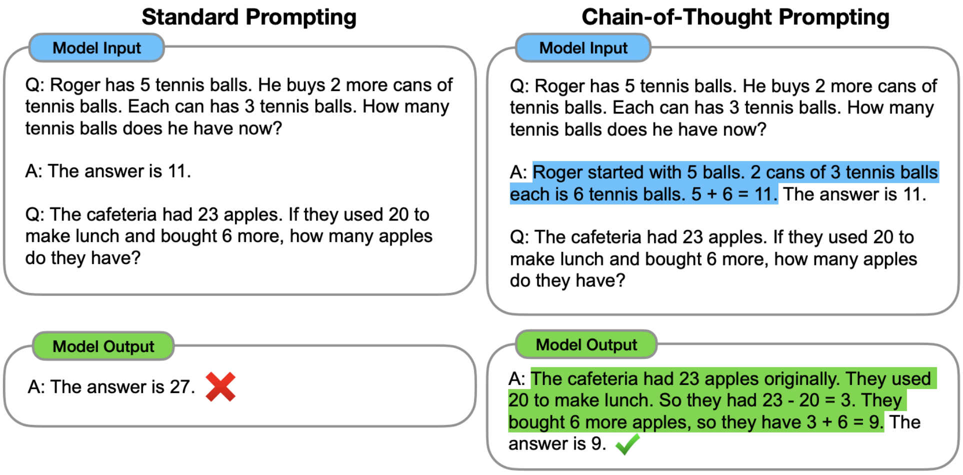

Figure 2.1: Example of Chain-of-Thought prompting used for multi-step reasoning. Source: Wei et al. (2022)

This method is equally applicable to logical problems like Zebra puzzles. Instead of asking "Who owns the zebra?" directly, the model is prompted to deduce one fact at a time, e.g., "The Norwegian lives in the first house. The house next to then must be blue. . . "

Multi-shot Prompting

Multi-shot prompting is a few-shot learning approach in which the prompt includes several examples of input–output pairs before the target query. It extends the few-shot prompting idea popularized by Brown et al. in GPT-3 Brown et al. (2020), and has been shown to improve consistency and accuracy, particularly in structured or domain-specific tasks.

For example, if a model is expected to generate structured JSON for a Zebra puzzle constraint, the prompt might include multiple examples of input-output pairs:

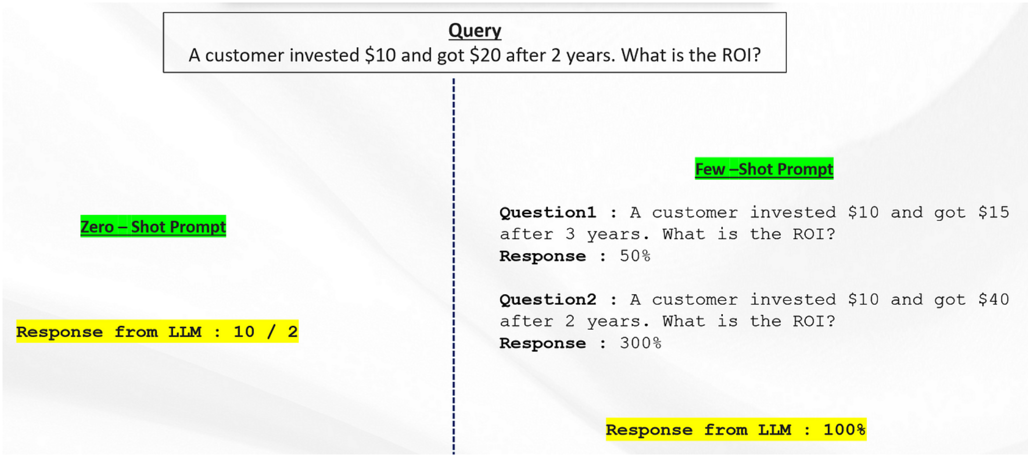

Figure 2.2: Example of Multi-shot prompting. Source: Talwar (2024)

This anchors the model's interpretation to a fixed schema and helps avoid hallucinations or inconsistent formatting. This is particularly critical when outputs must be programmatically parsed—such as when feeding them into the Z3 solver.

Together, CoT and multi-shot prompting form the foundation of the prompting strategies employed in this dissertation. They are evaluated in both reasoning (puzzle-solving) and formalization (constraint conversion) tasks to determine whether these techniques enhance LLM performance across the linguistic-functional spectrum.

2.2 Closely-Related Work

This section highlights research efforts closely aligned with the aims of this dissertation. Two works are particularly relevant: ZebraLogic: On the Scaling Limits of LLMs for Logical Reasoning Lin et al. (2025), which investigates the limits of LLMs in logical reasoning at scale, and the system proposed by Berman et al. Berman et al. (2024) in the paper: Solving Zebra Puzzles Using Constraint-Guided Multi-Agent Systems, which leverages multi-agent constraint feedback for solving Zebra puzzles. Each explores different dimensions of LLM capabilities in structured reasoning.

2.2.1 Evaluation Granularity and Scale

Lin et al. offer a large-scale evaluation of LLM performance on logic puzzles across a range of model sizes, prompt formats, and puzzle difficulties. It reveals the sharp decline in performance as puzzle complexity grows and finds that even advanced LLMs (e.g., GPT-4) struggle with symbolic constraints when not explicitly guided. However, the paper only evaluates direct solving and does not consider constraint translation as a separate linguistic task.

In contrast, this dissertation distinguishes between solving and converting, introducing cell-level accuracy and formal constraint matching as metrics. This enables a finer-grained assessment of LLM behavior and supports the core hypothesis around linguistic vs. functional competence.

2.2.2 Constraint Feedback and Interactive Reasoning

Berman et al. propose a constraint-guided, multi-agent system that tightly couples LLMs with an SMT solver through iterative feedback. Their architecture attempts to recover from conversion failures by incorporating solver errors as part of the reasoning loop, gradually steering the model toward a satisfiable formulation. This offers a promising paradigm for integrating symbolic validation into generation.

In comparison, this dissertation evaluates LLMs in a static, single-pass pipeline—without runtime feedback from the solver. Nevertheless, Section 4.4 discusses how errors encountered during Z3 execution were logged and how future systems might adopt feedback-based repair mechanisms similar to those used by Berman et al.

Summary

This chapter provided the theoretical and empirical foundations that contextualize the work presented in this dissertation. The background sections introduced the key pillars of this research: the evolution of Large Language Models (LLMs), the distinction between linguistic and functional competence, the formulation of Zebra puzzles as structured logic benchmarks, and the use of the Z3 solver as a formal verification backend. These concepts form the methodological basis for investigating how well different LLMs can reason under structured logical constraints.

The chapter also introduced and analyzed two key prompt engineering strategies—Chain-of-Thought and multishot prompting—which are tested throughout the dissertation as potential ways to enhance model performance. These methods, discussed in both conceptual and empirical terms, demonstrate how prompting alone can modulate the output fidelity and reasoning depth of language models in black-box settings.

The final section contrasted this dissertation's contributions with two closely related research efforts: The work of Lin et al. (2025), which focuses on large-scale benchmarking of LLMs on logic puzzles, and the work of Berman et al. (2024), which integrates symbolic solvers with LLMs through a feedback loop. While both projects offer valuable insights into the reasoning abilities of LLMs, this dissertation distinguishes itself by:

- Separating linguistic (constraint translation) and functional (puzzle solving) tasks for clearer competence attribution;
- Conducting controlled experiments across prompt strategies, puzzle difficulties, and model types;
- Evaluating the trade-off between accuracy and token efficiency, which is essential for real-world LLM deployment.

In doing so, this chapter not only lays the groundwork for the subsequent experimental analysis, but also positions this research within the broader discourse on LLM reasoning—highlighting both its novelty and relevance as a comprehensive diagnostic of modern language model behavior.

Work	Evaluation Scope	Use of Prompt Engineering
Lin et al.	Large-scale evaluation of LLM puzzle-solving performance with no emphasis on constraint translation or solver integration	Uses fixed prompts across models; does not explore prompt variants like CoT or multishot
Berman et al.	Enhances solving accuracy by tightly coupling LLMs with SMT solvers for iterative feedback-driven correction; does not examine linguistic vs. functional task separation	Relies on basic natural language instructions; does not investigate prompt variants or structured reasoning strategies
This Dissertation	Directly compares solving and constraint translation tasks using multiple LLMs and controlled prompts; evaluates both competence and efficiency trade-offs	Systematically evaluates baseline, Chain-of-Thought, and multishot prompting strategies across tasks and models

Table 2.2: Comparison of related research with this dissertation across evaluation scope and use of prompt engineering.

Chapter 3

Design

This chapter outlines the design of the experimental framework developed to investigate the comparative linguistic and functional competence of Large Language Models (LLMs). The primary aim of this design is to enable a controlled, interpretable evaluation of both capabilities using the same underlying task: solving Zebra puzzles.

To support this, a dual-pathway evaluation structure was devised. In the first path, the LLM is prompted to solve the puzzle directly in natural language, thereby testing its functional reasoning abilities. In the second, the LLM is tasked with converting the puzzle's natural language clues into a formal constraint representation, which is then executed by the Z3 constraint solver. This second path isolates the model's linguistic competence—its ability to extract and represent logical structure accurately from text.

The chapter begins by formulating the core problem addressed in this research and describing the specific limitations observed in prior work. It then details the primary challenges encountered in the design process, such as developing a custom constraint schema, defining reliable evaluation metrics, curating a benchmark dataset, and managing prompt engineering for multiple prompting strategies.

The proposed approach is then introduced, explaining how this dual-task framework allows for direct comparison between the two competencies while controlling for dataset and task variability. Finally, the overall system design is described, including data flow, prompt generation, model interaction, constraint evaluation, and the integration of structured prompting techniques such as Chain-of-Thought and Multi-shot prompting.

Together, these design choices establish the foundation for a robust and interpretable experimental setup capable of probing the relationship between language and reasoning in modern LLMs.

3.1 Problem Formulation

The central problem addressed in this dissertation is the evaluation of the reasoning capabilities of Large Language Models (LLMs) through the lens of two distinct cognitive competencies: linguistic competence and functional competence. While recent advances have led to significant improvements in natural language generation and instruction following, a growing body of research suggests that these models continue to exhibit notable deficiencies in structured reasoning tasks. This raises the question of whether LLMs are fundamentally better at handling linguistic surface forms than they are at performing the underlying reasoning those forms may represent.

As outlined in the previous chapter, linguistic competence refers to a model's ability to parse, reformat, and generate language in syntactically and semantically appropriate ways. Functional competence, by contrast, refers to the model's ability to carry out multi-step reasoning processes, maintain logical consistency, and solve constraint-based problems. Although many LLMs display fluency and coherence at the surface level, studies have shown a consistent pattern of failure in logical tasks such as Zebra puzzles, where performance remains far below human levels even in state-of-the-art models.

The challenge lies in isolating and evaluating each form of competence in a controlled setting. Most benchmarks conflate the two, making it difficult to determine whether a failure is due to an inability to understand the language or an inability to reason with it. This dissertation addresses this problem by designing a dual-pathway evaluation framework: one that requires the model to directly solve Zebra puzzles through natural language reasoning, and another that tasks the model with translating puzzle clues into a formal representation suitable for symbolic constraint solving. By evaluating performance in both tasks separately, this approach enables a clearer assessment of where current LLMs excel and where they fall short.

This formulation moves beyond general questions of model performance to focus on the more precise issue of competency dissociation. It seeks not just to measure success in completing tasks, but to identify whether the success stems from linguistic pattern-matching or true functional understanding. In doing so, it provides a foundation for both diagnosing current limitations and guiding future improvements in model design and evaluation methodology.

3.1.1 Identified Challenges

This dissertation aims to evaluate both linguistic and functional competence in LLMs using a shared task framework built around Zebra puzzles. While this unified approach creates a strong basis for comparison, implementing it required overcoming several key challenges. These challenges span from dataset construction and schema design to prompt engineering and evaluation methodology, each of which is detailed below.

Comparability Across Competencies

Most prior research isolates linguistic and functional competencies when evaluating LLMs. Linguistic abilities are typically assessed using tasks such as instruction following or formatting adherence, while functional reasoning is evaluated using logical puzzles, arithmetic challenges, or commonsense benchmarks. These tasks are rarely applied to the same problem instance, and are often drawn from distinct datasets, making direct comparisons between the two forms of competence difficult and potentially misleading.

This dissertation addresses that limitation by applying both evaluation modes—direct puzzle solving and natural language-to-logic constraint conversion—to the same set of Zebra puzzles. This design ensures that both competencies are tested under the same constraints, with identical input data. As a result, this approach reduces the confounding effects of task variation and enables a more interpretable, head-to-head analysis of linguistic versus functional performance in LLMs.

Designing a Custom Constraint Format

Zebra puzzles lack a standardized schema for encoding their logical constraints in a machine-readable format. Unlike other domains with established formal representations, there is no canonical encoding of Zebra puzzle clues that can be directly used with solvers like Z3. To enable symbolic evaluation, I designed a custom JSON-based intermediate format tailored for this task. However, this introduced new challenges: a single clue can be represented in multiple logically equivalent ways under Z3, leading to ambiguity in both generation and evaluation.

Consider the clue: *"The man who owns the dog lives next to the man who drinks tea."* This clue can be encoded using several logically equivalent constraint types:

- A `neighbor` constraint that explicitly enforces adjacency.
- An `abs_diff` constraint with a difference of 1.
- A combination of two separate directional constraints like `ImmediateLeft` and `ImmediateRight`, if both directions are accounted for.

This flexibility made it difficult to define a unique ground truth for comparison and increased the complexity of evaluating constraint accuracy.

Evaluating Constraint Translation

Evaluating the LLM's ability to solve puzzles directly was relatively straightforward—its solution could be compared against the known ground truth using both full-puzzle and per-cell accuracy. However, evaluating the quality of constraint conversion was significantly more difficult. If an LLM produced constraints that were 99% correct but included a minor syntax error, Z3 would fail entirely, leading to an all-or-nothing evaluation outcome. On the other hand, directly comparing the generated constraints against a manually defined reference set proved unreliable due to the multiple valid ways of encoding a clue.

To overcome this, I considered enforcing a restricted constraint vocabulary to guarantee uniformity. However, this simplified evaluation led to higher rates of hallucination—LLMs frequently generated unsupported or invented constraint types when limited in their expression. As a result, the final evaluation strategy for constraint conversion focused on whether the output, when passed to Z3, yielded the correct solution. Although imperfect, this approach provided a pragmatic balance between strictness and flexibility, given the rarity of syntactic errors.

Curating a Benchmark Dataset

Another challenge was the absence of an existing benchmark dataset pairing Zebra puzzles with both natural language clues and formal ground-truth constraint encodings. I manually constructed such a dataset by collecting puzzles of varying difficulty, rewriting their clues into a standard format, and creating the expected outputs for both solving and formalization. This required significant effort in aligning the puzzle logic with the custom constraint schema and validating the correctness of both puzzle solutions and their logical representations.

[*Note: The Zebra puzzles were sourced from a Hugging Face dataset, which provided raw puzzle clues and partial metadata. These were reformatted and extended manually. Full implementation details are discussed in the Implementation chapter.*]

Prompt Engineering for Robustness

Prompt engineering played a critical role in shaping the performance of the LLMs, especially in constraint conversion. Initial prompt variants frequently resulted in hallucinated constraint types, incorrect variable mapping, or mixing of explanation text into the structured output. For example, when using Chain-of-Thought prompting, LLMs would often embed explanatory reasoning inside the JSON solution field instead of isolating it under a designated key.

Numerous iterations were required to refine the prompts. This included clarifying the structure of expected outputs, explicitly listing supported constraint types, and separating reasoning from final answers. These refinements reduced hallucination rates and improved overall consistency, but also revealed model-specific sensitivities—some models required more rigid formatting than others to produce reliably parseable outputs.

Handling Edge Cases and Failures

Finally, implementing robust handling of edge cases was essential for maintaining pipeline stability. This included:

- Detecting malformed JSON outputs or missing required fields.
- Logging failed Z3 executions due to semantic or structural errors.
- Identifying cases where the LLM partially followed the schema but introduced incorrect or ambiguous logic.

Additionally, some Zebra puzzle clues were inherently difficult to formalize within the constraints of the schema. Handling such exceptions required either extending the format to accommodate rare structures or excluding those puzzles from the test set entirely.

3.1.2 Proposed Work

This dissertation proposes an empirical evaluation of the hypothesis that Large Language Models (LLMs) exhibit stronger linguistic competence than functional competence, as articulated in prior work by Mahowald et al. Mahowald et al. (2024). To test this, a two-pronged experimental framework is designed, wherein the same Zebra puzzle is presented to an LLM in two distinct modes of interaction:

1. **Direct Reasoning (Functional Competence)**: The LLM is tasked with directly solving the Zebra puzzle given in natural language, without any external symbolic tools or scaffolding.

2. **Constraint Translation (Linguistic Competence)**: The same LLM is asked to convert the natural language clues of the puzzle into a structured JSON format compatible with a custom interface to the Z3 constraint solver. This allows a symbolic reasoning system to compute the correct solution, and the accuracy of the LLM's translation is evaluated through the solver's results.

The dual-use of the same puzzle instance across both tasks offers a controlled and aligned benchmark for comparative analysis. Unlike prior studies that evaluate these competencies on separate datasets or domains, this shared testbed minimizes discrepancies in complexity, structure, and domain familiarity—allowing for a more robust interpretation of LLM performance across competencies.

To further assess and potentially enhance LLM performance, the proposed work integrates advanced *prompt engineering techniques*, specifically:

- **Chain-of-Thought (CoT) Prompting**: Encourages step-by-step natural language reasoning by the model prior to output generation, aimed at surfacing latent reasoning pathways that would otherwise remain implicit.

- **Multi-shot Prompting**: Provides multiple example input-output pairs as part of the prompt to guide the model's format adherence and semantic interpretation, especially useful in the constraint translation task.

These techniques will be evaluated in both the reasoning and conversion contexts to determine their efficacy in enhancing model performance. Their inclusion not only contributes to the main experimental objective but also serves to explore how prompt structure influences task-specific outcomes in reasoning-intensive applications.

Overall, the proposed framework combines structured logical tasks with interpretability-driven evaluation metrics to examine whether recent advances in LLM architecture and prompting methodology have meaningfully closed the gap between linguistic fluency and genuine reasoning capability.

3.2 Overview of the Design

To operationalize the comparative framework described in the previous section, a modular and extensible system was designed to evaluate both the reasoning and linguistic capabilities of Large Language Models (LLMs) using a consistent set of logic puzzles. The system architecture, depicted in Figure 4.1, reflects a clear separation of responsibilities across distinct components: input processing, LLM interfacing, constraint solving, and evaluation.

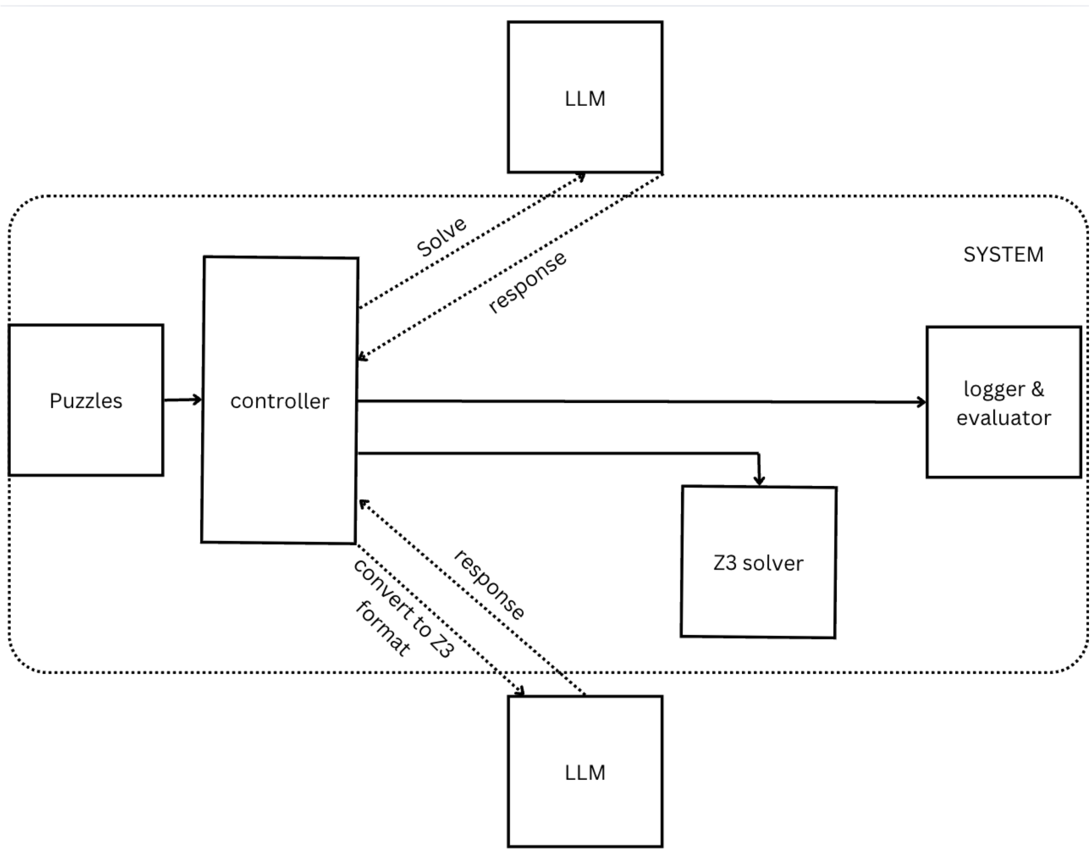

Figure 3.1: System architecture showing dual LLM pathways for solving and constraint translation.

The core controller ingests structured puzzle data and dispatches each instance along two processing paths:

1. In the **reasoning path**, the controller generates a solving prompt and sends it to the LLM. The LLM responds with its predicted full solution to the Zebra puzzle. This is then parsed and sent to the logging and evaluation modules for comparison with the known ground truth.

2. In the **translation path**, the same puzzle is fed into a different prompt that instructs the LLM to convert the clues into a structured JSON format compatible with a custom Z3 backend. The controller forwards this constraint representation to the solver, whose output is then logged and evaluated against the expected puzzle solution.

All results and metadata are recorded using a dedicated logging and evaluation module that captures performance metrics, error types, constraint validity, and execution traces. This setup enables reproducible experimentation and fine-grained comparative analysis across models and prompt types.

Prompt Engineering Integration

Given the high sensitivity of LLM outputs to input phrasing, prompt engineering forms a critical part of the system design. Three distinct prompting strategies were incorporated:

- **Zero-shot**: Standard instruction-only format with no examples or reasoning steps. Used as a baseline.

- **Chain-of-Thought (CoT)**: Prompts that explicitly encourage intermediate reasoning steps. Used in both solving and (to a limited extent) conversion pathways to test the impact of reflective inference on accuracy.

- **Multi-shot**: Prompts seeded with multiple input-output examples to anchor the expected response format. Primarily used in the constraint translation track to reduce syntactic errors and hallucinated constraint types.

These prompting styles were implemented through a dynamic prompt template system, allowing for modular experimentation across LLM types and runs. Prompt variations were carefully tuned to reduce format drift, constraint hallucination, and verbosity. For instance, when using CoT in conversion, it was essential to restrict reasoning to dedicated ‘explanation‘ keys rather than embedding logic into constraint outputs, which could corrupt downstream parsing.

Together, the system provides a flexible and extensible platform for evaluating reasoning and formalization capabilities in LLMs, isolating each component's contribution and enabling a controlled comparison between linguistic and functional competence.

3.3 Summary

This chapter outlined the conceptual and methodological foundations upon which the experimental framework of this dissertation is built. It began by formally defining the problem of evaluating linguistic versus functional competence in Large Language Models (LLMs), motivated by observed limitations in prior work that tend to examine these competencies in isolation and across non-comparable task settings.

The chapter then described a set of concrete challenges encountered during the design process. These included: the need for a unified testbed that supports both solving and constraint formalization; the absence of standardized Z3 formats for Zebra puzzles; the complexity of creating ground truth datasets; and the difficulty of fairly evaluating constraint translation performance due to representational variance. Additionally, prompt engineering was highlighted as a non-trivial aspect of the design, requiring careful experimentation to mitigate hallucinations and enforce adherence to schema requirements.

To address these challenges, the proposed solution leverages a dual-evaluation strategy applied to the same Zebra puzzles: direct puzzle-solving to assess functional reasoning, and structured constraint translation to assess linguistic competence. Advanced prompting strategies—Chain-of-Thought and Multi-shot prompting—were incorporated into the framework to probe whether structured prompting can enhance performance across both tasks.

Together, these design decisions establish a robust and interpretable experimental platform for evaluating the core hypothesis of this dissertation. The next chapter will detail how this design is implemented in practice, including the software architecture, data pipeline, and execution logic of the system.

Chapter 4

Implementation

This chapter presents the implementation of the experimental framework designed to evaluate the comparative linguistic and functional competence of Large Language Models (LLMs) using Zebra puzzles. Building upon the architectural design outlined in the previous chapter, the system was implemented as a modular and extensible pipeline that automates the full evaluation processfrom input preprocessing and prompt generation to model interaction, constraint solving, and results evaluation.

The chapter begins with an overview of the system architecture and overall execution flow. It then provides a detailed breakdown of each major component, including the puzzle processing module, the prompt generator, the LLM interface layer, the Z3 solver integration, and the evaluation and logging infrastructure. Where appropriate, key implementation decisions, challenges encountered, and illustrative code examples are included to clarify the internal workings of the system.

The remainder of this chapter explains how these components were implemented and integrated into a functioning system.

4.1 Overview of the Solution

The system was implemented as a modular Python-based pipeline designed to reflect the dual-path evaluation structure introduced in the design. Its purpose is to process Zebra puzzles through two distinct execution routesdirect reasoning and constraint translation-while unifying them under a shared evaluation and logging framework.

The architecture of the system is shown in Figure 4.1. It illustrates how structured puzzle inputs flow through the two main processing pathways. One path directs prompts to the LLM for solving puzzles using natural language reasoning. The other path prompts the LLM to convert clues into a formal constraint representation, which is then handled by a symbolic solver, i.e z3. Both outputs are evaluated using a common evaluation module that supports accuracy tracking, logging, and performance comparison.

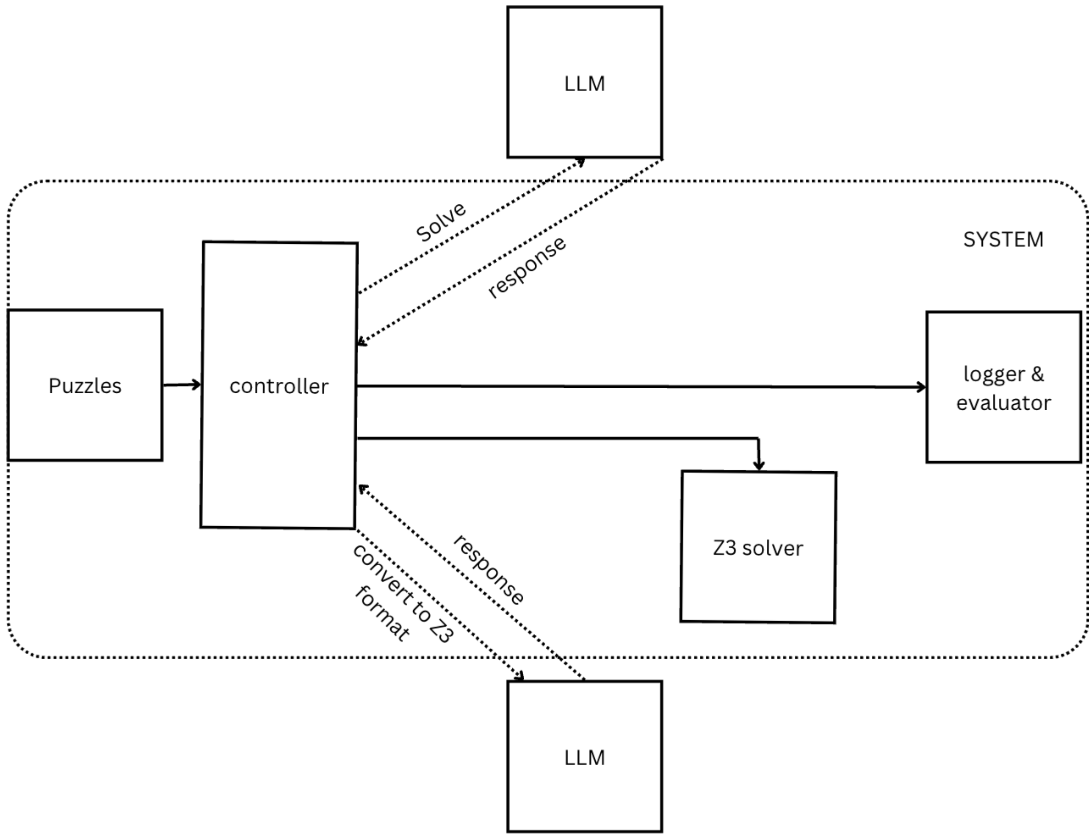

Figure 4.1: System architecture showing dual LLM pathways for solving and constraint translation.

The core orchestration logic of the system is handled by the controller module, implemented in `main.py`. This module serves as the entry point for the program and is responsible for interpreting the command-line arguments provided by the user. These arguments determine which LLM (OpenAI, DeepSeek, or Mistral) should be used, which puzzles should be processed (a specific one or all available), whether the system should execute solving, conversion, or both tasks, and which prompting strategy to apply (e.g., zero-shot, chain-of-thought, or multi-shot). Based on these parameters, the controller initializes the appropriate model and evaluation environment, including the Z3 solver if constraint conversion is involved.

Once initialized, the controller iterates through the specified puzzle or set of puzzles. For each one, it constructs the corresponding prompt by calling the `prompt_generator.py` module, which builds task-specific instructions aligned with the selected prompting strategy. The LLM is then queried with this prompt and returns its response, along with metadata such as token usage, latency, and any error messages. If the response contains structural issuessuch as malformed JSON or incorrectly formatted fieldsbasic correction logic is applied before proceeding.

In the case of a conversion task, the cleaned LLM output is passed to the `z3_solver.py` module, which parses the constraint representation and attempts to compute a solution. The resulting output, whether successful or failed, is handed off to the `logger.py` module. This module compares the predicted solution to the ground truth, evaluates both full and per-cell accuracy, and logs the results to disk in a structured format. After all puzzles have been processed, the `benchmark.py` script can be used to aggregate and summarize the log data into performance reports, enabling analysis of model performance across tasks, prompting strategies, and puzzle difficulty levels.

4.2 Component One: Puzzle Dataset and Representation

The `puzzle.py` file defines the full suite of logic puzzles used in the evaluation framework. These puzzles serve as input to both the solving and constraint-translation branches of the system and are encoded in a structured JSON-like format to ensure consistent processing across modules. In total, 50 puzzles were included in the final dataset.

All puzzles, except for the well-known five-house version popularized in *Life International* in 1962 and cited from Wikipedia Wikipedia contributors (2025), were sourced from the publicly available ZebraLogic dataset published on Hugging Face Lin et al. (2025). This benchmark is one of the largest standardized evaluations for LLM-based logic reasoning. Each puzzle entry from the dataset was manually reformatted and rewritten to align with the specific input-output interface of this system, ensuring compatibility with both the prompt generation and Z3-based evaluation components.

id string · lengths 14 15	size string · classes 25 values	puzzle string · lengths 394 2.95k	solution dict	created_at string · lengths 26 26
lgp-test-5x6-16	5*6	There are 5 houses, numbered 1 to 5 from…	{ "header": ["House", "Name", "Nationality"…	2024-07-03T21:21:29.209499
lgp-test-4x4-27	4*4	There are 4 houses, numbered 1 to 4 from…	{ "header": ["House", "Name", "Occupation",…	2024-07-03T21:21:29.207505
lgp-test-6x4-15	6*4	There are 6 houses, numbered 1 to 6 from…	{ "header": ["House", "Name", "Children",…	2024-07-03T21:21:29.210360
lgp-test-6x5-2	6*5	There are 6 houses, numbered 1 to 6 from…	{ "header": ["House", "Name", "Mother",…	2024-07-03T21:21:29.210554
lgp-test-2x2-33	2*2	There are 2 houses, numbered 1 to 2 from…	{ "header": ["House", "Name", "Pet"],…	2024-07-03T21:21:29.204640
lgp-test-4x4-3	4*4	There are 4 houses, numbered 1 to 4 from…	{ "header": ["House", "Name", "Mother",…	2024-07-03T21:21:29.207309

‹ Previous 1 2 3 ... 10 Next ›

Figure 4.2: Sample entries from the Hugging Face dataset.

Each puzzle dictionary contains:

- `text_description`: A natural language version of the puzzle as it would be presented to an LLM.
- `size`: A string denoting the grid structure (e.g., `5x5`, `3x2`).
- `z3_format`: (Optional) A structured representation with categories and constraints, used to validate constraint translation. Present only in the first 10 puzzles.
- `ground_truth_dict`: The correct answer for each entity, mapping category items to house positions.

Notably, the `z3_format` field was originally included to serve as a reference for evaluating constraint translation accuracy. However, due to challenges described in Section 3.1.1such as ambiguity in how clues can be formally encoded, the need for strict formatting, and the substantial manual effort required to author consistent ground truthsthis field was only constructed for the first 10 puzzles. For the remaining 40 puzzles, only solution-level evaluation is performed.

Puzzle Distribution by Size Category

Following the classification schema proposed by the ZebraLogic benchmark Lin et al. (2025), the 50 puzzles are categorized as follows:

- **Small**: 19 puzzles (e.g., 2x3, 3x2, 4x2)
- **Medium**: 15 puzzles (e.g., 4x4, 3x5, 6x2)
- **Large**: 7 puzzles (e.g., 5x4, 4x5, 6x3)
- **Extra-Large**: 9 puzzles (e.g., 5x5, 6x4, 6x6)

This range ensures diversity in reasoning complexity and allows systematic evaluation of model behavior across different puzzle sizes.

Example 1: Structured 5x5 Puzzle with Constraints


```
"puzzle_1": {
  "text_description": "There are five houses. The Englishman...",
  "size": "5x5",
  "z3_format": {
    "houses_count": 5,
    "categories": {
      "nationalities": ["Englishman", "Spaniard", "Ukrainian"..],
      "colors": ["Red", "Green", "Ivory", ...],
      ...
    },
    "constraints": [
      {"type": "eq", "var1": "Englishman", "var2": "Red"},
      {"type": "eq_offset", "var1": "Green", "var2": "Ivory"...},
      {"type": "neighbor", "var1": "Kools", "var2": "Horse"},
      ...
    ]
  },
  "ground_truth_dict": {
    "Englishman": 3,
    "Green": 5,
    ...
```

```
  }
}
```

Listing 4.1: Structured JSON-format representation of a 5x5 Zebra puzzle including category and constraint metadata.

This format is used for puzzles where both solving and constraint translation tasks are performed.

Example 2: Structured 3x5 Puzzle Without Constraints


```
"puzzle_19": {
  "text_description": "There are 3 houses, numbered 1 to 3...",
  "size": "3x5",
  "ground_truth_dict": {
    "Eric": 1,
    "red": 1,
    "city": 1,
    "sept": 1,
    "Fred": 1,
    "Peter": 2,
    "yellow": 2,
    "mountain": 2,
    "jan": 2,
    "Bella": 2,
    "Arnold": 3,
    "white": 3,
    "beach": 3,
    "april": 3,
    "Meredith": 3
  }
}
```


Listing 4.2: Puzzle used for solving-only tasks due to missing structured constraint annotation.

This example typifies the 40 puzzles used for end-to-end reasoning evaluation only. These puzzles omit `z3_format` fields and are assessed solely based on the correctness of their final answer mappings.

In summary, `puzzle.py` functions as both the definition and validation source for each logical reasoning task in the experiment. Its design supports modularity and scalability while maintaining alignment with benchmarking best practices.

4.3 Component Two: Controller and Prompt Generator

At the heart of the system lies the `main.py` file, which serves as the controller module responsible for orchestrating the full pipeline. It interprets user-specified arguments from the command linesuch as which LLM to use (OpenAI, DeepSeek, or Mistral), which puzzle(s) to process, whether to perform solving, constraint translation, or both, and what prompt strategy to use. Based on these parameters, the appropriate model interfaces and support components are initialized.

System Execution Flow

Once initialized, the controller reads the specified puzzle(s) from the dataset and executes the corresponding task(s). If solving is enabled, the controller queries the LLM with a generated prompt (constructed via `prompt_generator.py`) to obtain a direct solution. If constraint conversion is required, it similarly queries the LLM to produce a structured Z3-compatible format. In both cases, the output is cleaned, parsed, and stored, and if applicable, passed to the `z3_solver.py` module for validation. Once all responses are processed, the controller forwards the final outputs and metadata to the `logger.py` module for structured evaluation and logging.

Prompt Strategy Design

The `prompt_generator.py` module contains the function `get_prompt()`, which dynamically returns a formatted instruction prompt based on the selected task mode (solve/-convert) and strategy (baseline, CoT, or multi-shot). Each strategy tailors the prompt to guide the LLM's behavior in distinct ways.

Baseline Prompting

The baseline strategy is the simplest formulation. It directly instructs the model to provide a JSON dictionary containing the final solution, without explanation or scaffolding.

This method reflects standard prompting as used in early LLM benchmarks like GPT-3's original few-shot demonstrations Brown et al. (2020).

```
Solve this puzzle by returning a Python/JSON dictionary
that maps each distinct item (like colors or names) to an integer house index,
where House #1 is the leftmost and House #N is the rightmost.

No explanations or commentary, just the dictionary.

For example, if there are two items: Red, Blue for houses left to right,
and we know Red=1 (left), Blue=2 (right). If the puzzle also says
PersonA=1, PersonB=2, the final dictionary is exactly:

{
  "Red": 1,
  "Blue": 2,
  "PersonA": 1,
  "PersonB": 2
}

No extra text, just that dictionary structure.
```

Listing 4.3: Prompt format for direct dictionary output with no intermediate reasoning.

For conversion, the baseline prompt simply asks for a valid Z3 schema without including any explanation or reasoning:

```
You are given a logic puzzle.

Convert this puzzle into a JSON structure usable by a Z3 solver:
- The JSON must have: "houses_count", "categories", and "constraints".
- "categories" is a dictionary mapping each category name to a list of items (strings)
    .
- "constraints" is a list of objects describing each constraint. Each constraint must
    use exactly one of the following types:
  - "eq"
  - "eq_offset"
```

– "neighbor"
– "neq"
– "ImmediateLeft"
– "ImmediateRight"
– " distinct_categories "
– "range"
– "rightOf"
– "leftOf"
– "abs_diff"

– Use only the following keys as needed: "type", "var1", "var2", " offset ", "var2int", " diff ", **and** "categories".

– For " distinct_categories ", ALWAYS use the **format**:
{
"type": " distinct_categories ",
" categories ": [" colors ", "names", ...]
}
(Do **not** include "var1" **or** "var2" **in** this constraint .)

– Include exactly one "range" constraint:
{
"type": "range",
"from": 1,
"to": 5
}

– Do NOT invent new constraint types **or** items that are **not** part of the puzzle's categories.
– Do NOT include explanations, reasoning, or extra text . Only output valid JSON.
– If referencing a numeric house index in a constraint (e.g., neq), use "var2int":
```
  {
    "type": "neq",
    "var1": "X",
    "var2int": 2
  }
```

Example output (dummy puzzle, not your real puzzle):

```
{
  "houses_count": 5,
  "categories": {
    "colors": ["Red", "Green", "Blue", "Yellow", "White"],
    "people": ["Alice", "Bob", "Charlie", "Diana", "Evan"]
  },
  "constraints": [
    {
      "type": "distinct_categories",
      "categories": ["colors", "people"]
    },
    {
      "type": "range",
      "from": 1,
      "to": 5
    },
    { "type": "eq", "var1": "Alice", "var2": "Red" },
    { "type": "eq_offset", "var1": "Blue", "var2": "Green", "offset": 1 },
    { "type": "neighbor", "var1": "Bob", "var2": "Alice" }
  ]
}

ONLY output valid JSON. No commentary.
Puzzle:
```

Listing 4.4: Prompt format instructing the model to output a valid constraint schema with no commentary.

Chain-of-Thought Prompting

Chain-of-Thought (CoT) prompting is designed to encourage the model to externalize its reasoning process by producing a step-by-step explanation before delivering the final output Wei et al. (2022). This strategy aims to improve model performance on tasks involving multi-step logic, and is particularly relevant for Zebra puzzles due to their deep constraint dependencies.

In solving mode, the prompt asks the LLM to first articulate its reasoning in an `"explanation"` field, then provide the structured solution dictionary.

```
"""Think critically and show your step-by-step reasoning in the "explanation" field to
    solve the puzzle below.
Then produce a final JSON with exactly these two fields:

{
  "explanation": "...",
  "solution": { ... }
}

**Do not** use triple-backticks or any code fences.
**Do not** include any extra text outside the JSON.
**Do not** add Markdown headings or bullet points.
Your entire response must be valid JSON, period.

Below are the rules for solving:
(Same as the baseline instructions)
"""
```

Listing 4.5: Full instructional content for guiding CoT-based reasoning followed by a structured answer.

In conversion mode, CoT prompting also includes an `"explanation"` field alongside the structured Z3 output under a `"z3"` key.

```
"""Think critically and, show your step-by-step chain-of-thought reasoning (
    explanation).
Then, produce a final JSON with two keys: "explanation" for your reasoning, and
"z3" for the actual constraints.

Below are the rules for converting puzzles to Z3:
(Same as the baseline instructions)
"""
```

Listing 4.6: Prompt guiding the model to reason through constraint translation and return a valid Z3 schema.

This format enhances interpretability by capturing the models reasoning process alongside its final output.

Multi-shot Prompting

Multi-shot prompting builds on few-shot learning by including multiple exemplar prompts and their outputs before the target task. These demonstrations teach the model the expected response pattern via repetition.

In solving mode, multiple sample puzzles with their corresponding dictionary solutions are included before the target puzzle.

```
"""Solve this puzzle by returning a Python/JSON dictionary
that maps each distinct item (like colors or names) to an integer house index,
where House #1 is the leftmost and House #N is the rightmost. No explanations
or commentary, just the dictionary.

For example, for this question:
[Example Puzzle Text 1]

The Solution would be:
{
  "Englishman": 3,
  "Spaniard": 4,
  "Ukrainian": 2,
  "Norwegian": 1,
  "Japanese": 5,
  ...
}

For this question:
[Example Puzzle Text 2]

The solution would be:
{
  "Red": 1,
  "Blue": 2,
  "PersonA": 1,
  "PersonB": 2
}
```

```
For this question:
[Example Puzzle Text 3]

The solution would be:
{
  "Peter": 2,
  "Eric": 3,
  "Arnold": 1,
  "tea": 1,
  "water": 2,
  "milk": 3
}

No extra text, just that dictionary structure.
"""
```

Listing 4.7: Prompt format that includes multiple solved examples before presenting the target puzzle.

For conversion, the multi-shot prompt provides several examples of valid Z3-formatted constraints and schemas. The prompt ends by instructing the model to format the current puzzle similarly.

```
""You are given a logic puzzle.

Convert this puzzle into a JSON structure usable by a Z3 solver:
Below are the rules for converting:
(Same as the baseline instructions)

For example, for this question: [Puzzle Text 1]
The Solution would be:[Z3 Solution 1]

for this question: [Puzzle Text 2]
The Solution would be:[Z3 Solution 2]

for this question: [Puzzle Text 3]
The Solution would be:[Z3 Solution 3]
```

```
ONLY output valid JSON. No commentary.
Puzzle:
[Puzzle Text]
"""
```

Listing 4.8: Prompt format for converting text to Z3 constraints with multiple puzzle-solution pairs.

Robustness and Error Handling

Since even well-formatted prompts may result in noisy outputs, all responses are cleaned using a dedicated `clean_response()` function. This process strips Markdown, escapes characters in JSON, removes extraneous explanations from structured fields, and ensures proper parsing.

Final Remarks on Controller and Prompt Design

Together, the controller and prompt generation modules form a flexible and extensible core of the system. Their combined logic allows for modular experimentation with model types, puzzle complexity, and prompting styles. The structure of these components ensures reproducibility while supporting diverse strategies for reasoning and translation evaluation. The final results, along with key metadata such as response time, token usage, and any errors encountered, are then passed to the `logger.py` module. This module is responsible for formal evaluation and persistent logging of each run, setting the stage for comparative benchmarking across models and prompt strategies.

4.4 Component Three: LLM Solver Modules

The system supports three primary LLM providers-OpenAI, DeepSeek, and Mistral-each integrated through a dedicated solver module. These modules abstract away API differences and expose a unified interface to query the selected model using structured prompts, returning the raw response, response time, and token usage. The key methods implemented are `query_llm()`, `solve_puzzle()`, and `convert_to_z3_format()`.

Each provider module uses a consistent querying structure built around a conversational prompt schema. Specifically, messages passed to the model follow the format:

- A `system` message defining the task constraints (e.g., "You are an expert puzzle solver. Output only valid JSON with no extra commentary.").

- A `user` message containing the generated prompt (from `prompt_generator.py`) corresponding to either a solving or constraint conversion task.

Additionally, each solver sets the `temperature` parameterused to control the randomness of outputto a low value (e.g., 0.3) to ensure deterministic behavior during evaluation runs. This configuration is especially critical when evaluating prompt strategies such as Chain-of-Thought or Multi-shot prompting, where consistency and format adherence are key.

Supported Models

Across the three solvers, the following models were tested:

- **OpenAI**: `gpt-4o` and `o3-mini`
- **DeepSeek**: `deepseek-reasoner` and `deepseek-chat`
- **Mistral**: `mistral-small-latest`

Each model was evaluated across multiple puzzles and prompt strategies. The same solver interface was used regardless of provider, simplifying benchmarking and logging across LLM variants.

Robustness and Retry Mechanisms

To account for rate limits and transient API failures, each solver implements a retry loop with exponential backoff. For instance, in the DeepSeek and OpenAI solvers, failed requests (due to timeouts or parsing errors) are retried up to 10 times with increasing wait durations. This allowed for stable execution across batch runs involving dozens of queries per model.

Output Structure and Return Format

Each solver returns three items from a query:

- The **raw text response** from the LLM, which is parsed downstream.
- The **response time** in seconds, calculated as wall-clock time from submission to reply.
- The **token usage**, if available via the API's usage metadata; otherwise, `"N/A"` is logged.

These values are used in performance benchmarking, evaluation, and cost estimation. Structured outputs-such as the JSON-formatted `"solution"` or `"z3"` objects-are parsed and validated downstream, with failures caught and logged in `logger.py`.

Error Handling and Fallbacks

If the returned string fails to parse as valid JSON (a common issue with LLMs when under-constrained), the system logs the error but continues execution. For certain failure modes, such as malformed constraints, the solver may return `None`, triggering fallback routines or logging appropriate diagnostics.

This modular architecture allows future substitution or extension of LLM providers with minimal change to the rest of the pipeline.

4.5 Component Four: Z3 Constraint Solver

The `z3_solver.py` file implements the constraint solver backend responsible for executing formal reasoning once the puzzle has been converted into a valid Z3-compatible schema. This component serves as a verification mechanism for the linguistic competence task. If the constraints are well-formed, a solution should be derivable using symbolic logic alone.

The solver leverages the `Z3` Satisfiability Modulo Theories (SMT) engine De Moura and Bjørner (2008); Z3TheoremProver (2012), a mature and widely adopted tool developed by Microsoft Research, known for its robustness in declarative reasoning over logical formulas.

Architecture and Workflow

At the core is the `ZebraSolver` class, which accepts a parsed JSON representation of a puzzle (the same format expected from the LLM in the conversion task). This representation includes:

- `houses_count`: The number of positions (typically 3-6).
- `categories`: A dictionary mapping each attribute category (e.g., `colors`, `drinks`) to a set of distinct values representing possible assignments within that category.
- `constraints`: A list of constraint dictionaries representing relationships between items.

Upon initialization, the solver creates a unique `Int` variable in Z3 for each item in every category, and maps them internally in `self.item_vars`.

```
self.item_vars = {}
for cat_name, items in self.categories.items():
    for item in items:
        self.item_vars[item] = Int(item)
```

Listing 4.9: Variable setup and internal structure of solver state.

Constraint Handling

The solver supports a wide range of constraint types, covering positional equality, offset relations, adjacency, ordering, and absolute differences. It processes these constraints in two passes:

- **Pass 1**: Global constraints like `distinct_categories` and `range`, which apply across entire attribute groups.
- **Pass 2**: Local constraints between item pairs, such as `eq`, `neighbor`, `ImmediateLeft`, `abs_diff`, and others.

Each constraint type is parsed, validated, and translated into a Z3 logical expression. Invalid or unknown types are logged in `self.errors`.

```
elif ctype == "neighbor":
    var1 = c.get("var1")
    var2 = c.get("var2")
    if not var1 or not var2 or var1 not in self.item_vars or var2 not in self.
    ↪ item_vars:
        self.errors.append(f"neighbor referencing unknown items: {c}")
        continue
    self.solver.add(Or(
        self.item_vars[var1] == self.item_vars[var2] + 1,
        self.item_vars[var1] == self.item_vars[var2] - 1
```

Listing 4.10: Representation of neighbour constraint using index arithmetic.

This modular design makes the solver highly tolerant to syntactic errors while ensuring all valid constraints are incorporated.

Solving and Model Extraction

After constraint processing, the solver checks satisfiability using Z3s `check()` function. If satisfiable, it returns a dictionary mapping each entity to its house position as determined by the solvers internal model:

```
if result == sat:
    model = self.solver.model()
    solution = {}
    for it in self.item_vars:
        solution[it] = model[self.item_vars[it]].as_long()
    return solution
```

Listing 4.11: Reading a satisfying assignment from the Z3 model.

Robustness and Error Reporting

Rather than halting on the first malformed constraint, the solver accumulates errors throughout parsing. This makes it easier to debug LLM-generated constraint outputs by inspecting which specific clues were misrepresented.

If no solution is found (unsatisfiable constraints or inconsistent logic), or if critical constraints were missing, the solver returns `None`, and the issue is logged upstream for analysis.

While the current system treats the solver as a passive verifier, it could potentially serve an even greater role in future work. By feeding Z3-derived errors or unsatisfiability feedback back into the LLM, one could construct a reasoning feedback loop that encourages model correction and self-improvement. However, such iterative refinement mechanisms lie beyond the scope of this dissertation.

Final remarks on Z3 solver

The Z3 solver serves as the formal reasoning benchmark in this system. Its successful resolution of the constraint set validates the correctness of the LLM's linguistic output.

By isolating errors in the translation step from errors in logic execution, this module plays a central role in assessing the dissociation between linguistic and functional competence.

4.6 Component Five: Logging and Evaluation Framework

The `logger.py` module implements a comprehensive evaluation and logging framework that records and scores each model run. It tracks results across both the puzzle-solving and constraint-translation tasks, storing outputs, metadata, and performance metrics in a unified format. Logs are stored in `results/log.json` and later consumed by the benchmarking tool.

Evaluation Scope

The logger captures and scores outputs along two primary dimensions: **puzzle solving** and **constraint-based conversion**. Each of these is assessed using both *complete solution accuracy* and *cell-level accuracy*:

- **Solving Accuracy**: The model's direct answer is parsed as a dictionary and compared to the puzzle's ground truth. If all values match exactly, the solution is counted as fully correct. Additionally, a more granular metric is computed by measuring how many individual assignments (e.g., `"Red": 3`) are correctly placed.
- **Conversion Accuracy via Solver**: After converting the puzzle into a constraint format, the LLMs output is passed to the Z3 solver. The resulting numeric assignment is again compared against the ground truth using both full-match and cell-level scoring.
- **Constraint Format Matching (Limited)**: In a small subset of puzzles with official constraint annotations, the logger also attempts to compare the raw constraint structures generated by the LLM with the ground-truth Z3 format. However, this metric is subject to two significant limitations:
  - It is only available for the first 10 puzzles.
  - Many clues can be validly encoded in multiple semantically equivalent ways (e.g., using either `"eq_offset"` or `"ImmediateRight"`), which leads to mismatches despite logical correctness.

As such, constraint format accuracy is considered incomplete in the context of this project and its figures are not reported in the final analysis. Nevertheless, the mechanism remains conceptually valuable, as it provides a deeper lens into the models structural understanding. With further refinement, such evaluations could enable targeted insights into specific types of misrepresentations or systematic errors in constraint translation.

Explanation of Log Keys and Metrics

To facilitate reproducibility and structured evaluation, each logged run includes a comprehensive set of metadata and accuracy measures. The most relevant fields are described below:

- `timestamp`: The date and time at which the log was recorded.
- `llm_provider`: The large language model used for the run (`openai`, `deepseek`, or `mistral`).
- `puzzle`, `puzzle_size`: Identifier and dimensions of the logic puzzle.
- `variant`: Whether the run was a `solve`, `convert`, or `full_test` (both).
- `strategy`: Prompting strategy used: `baseline`, `cot`, or `multishot`.
- `chain_of_thought`: Explanation field returned during CoT runs; contains reasoning steps if available.
- `prompt`: The natural language prompt sent to the LLM.
- `solve_dict_str`: The models direct solution as a dictionary (string-encoded).
- `solve_parsed`: Parsed JSON version of the dictionary, or error message if parsing failed.
- `solve_accuracy`: The proportion of correctly assigned items compared to the ground truth (i.e., cell-level accuracy).
- `solve_correct_fields` and `solve_total_fields`: Raw numerator and denominator for the solve accuracy.
- `solve_response_time`, `solve_token_usage`: Time in seconds and token count used by the LLM to produce a solution.

- `convert_constraints`: The constraint-based output generated by the LLM (JSON string).
- `convert_solver_str`: Result of feeding the converted constraints into the Z3 solver; mirrors the ground-truth dictionary format if successful.
- `convert_solver_accuracy`, `convert_correct_fields`, `convert_total_fields`: Accuracy of the Z3-produced result using the same metric as in solving.
- `convert_solver_parsed`: Parsed dictionary version of the solvers output, or error message if parsing failed.
- `official_z3_constraints_present`: Boolean flag indicating whether a gold standard Z3 format was available for comparison.
- `constraints_accuracy`: Accuracy of the LLM-generated Z3 constraint structure against the official format (when applicable). Due to ambiguity and variability in constraint expression, this is not used as a definitive metric in this project.
- `error`: Any encountered issues during the run, including failed parsing, malformed output, or solver exceptions.

Example Log Entries

Each evaluation entry is written as a structured JSON object, including solving metrics, conversion accuracy, metadata such as token usage and timing, and any encountered errors. Below are two representative examples:


```
{
"timestamp": "2025-04-15 00:11:33",
"llm_provider": "deepseek",
"puzzle": "puzzle_44",
"puzzle_size": "2x3",
"variant": "full_test",
"strategy": "cot",
"chain_of_thought": "Solve: From clue 2, Arnold (1) is left of
    Eric (2). Clue 3 places Desert in house 1, so Cherry must be
    in 2. Clue 1 links Cherry to iPhone 13 (house 2), leaving
    Samsung S21 for house 1. Thus: Arnold=1, Eric=2; desert=1,
    cherry=2; samsung galaxy s21=1, iphone 13=2.; Convert: The
```

```
    puzzle involves 2 houses with categories: names (Arnold, Eric
    ), smoothies (desert, cherry), and phones (samsung galaxy s21
    , iphone 13). Clue 1 links cherry and iphone 13 with 'eq'.
    Clue 2 places Arnold immediately left of Eric using '
    ImmediateLeft'. Clue 3 ensures desert is not in house 2 via '
    neq' with var2int:2. 'distinct_categories' ensures uniqueness
     across all categories, and 'range' sets houses 1-2.",
"prompt": "There are 2 houses, numbered 1 to 2 from left to
    right, as seen from across the street. Each house is occupied
     by a different person. Each house has a unique attribute for
     each of the following characteristics:\n- Each person has a
    unique name: `Arnold`, `Eric`\n- Everyone has a favorite
    smoothie: `desert`, `cherry`\n- People use unique phone
    models: `samsung galaxy s21`, `iphone 13`\n\n## Clues:\n1.
    The person who likes Cherry smoothies is the person who uses
    an iPhone 13.\n2. Arnold is directly left of Eric.\n3. The
    Desert smoothie lover is not in the second house.",
"puzzle_ground_truth_dict": {
     "Arnold": 1,
     "Eric": 2,
     "desert": 1,
     "cherry": 2,
     "samsung galaxy s21": 1,
     "iphone 13": 2
},
"solve_dict_str": "{\"Arnold\": 1, \"Eric\": 2, \"desert\": 1,
    \"cherry\": 2, \"samsung galaxy s21\": 1, \"iphone 13\": 2}",
"solve_response_time": 23.71,
"solve_token_usage": 1075,
"solve_accuracy": 1.0,
"solve_correct_fields": 6,
"solve_total_fields": 6,
"solve_parsed": {
     "Arnold": 1,
     "Eric": 2,
     "desert": 1,
     "cherry": 2,
```

```
    "samsung-galaxy-s21": 1,
    "iphone-13": 2
},
"convert_constraints": "{\"houses_count\": 2, \"categories\": 
   {\"names\": [\"Arnold\", \"Eric\"], \"smoothies\": [\"desert
   \", \"cherry\"], \"phones\": [\"samsung-galaxy-s21\", \"
   iphone-13\"]}, \"constraints\": [{\"type\": \"
   distinct_categories\", \"categories\": [\"names\", \"
   smoothies\", \"phones\"]}, {\"type\": \"range\", \"from\": 1,
    \"to\": 2}, {\"type\": \"eq\", \"var1\": \"cherry\", \"var2
   \": \"iphone-13\"}, {\"type\": \"ImmediateLeft\", \"var1\": 
   \"Arnold\", \"var2\": \"Eric\"}, {\"type\": \"neq\", \"var1
   \": \"desert\", \"var2int\": 2}]}",
"convert_response_time": 48.81,
"convert_token_usage": 2451,
"convert_solver_str": "{\"Arnold\": 1, \"Eric\": 2, \"desert\": 
   1, \"cherry\": 2, \"samsung-galaxy-s21\": 1, \"iphone-13\": 
   2}",
"convert_solver_accuracy": 1.0,
"convert_correct_fields": 6,
"convert_total_fields": 6,
"convert_solver_parsed": {
    "Arnold": 1,
    "Eric": 2,
    "desert": 1,
    "cherry": 2,
    "samsung-galaxy-s21": 1,
    "iphone-13": 2
},
"official_z3_constraints_present": false,
"constraints_accuracy": 0.0,
"constraints_correct_fields": 0,
"constraints_total_fields": 1,
"error": "N/A"
},
```

Listing 4.12: An entry showing perfect solve and conversion from DeepSeek using Chain-of-Thought prompting.


```
{
"timestamp": "2025-04-14 01:50:21",
"llm_provider": "openai",
"puzzle": "puzzle_29",
"puzzle_size": "5x3",
"variant": "full_test",
"strategy": "baseline",
"chain_of_thought": "Solve: N/A; Convert: N/A",
"prompt": "There are 5 houses, numbered 1 to 5 ...",
"puzzle_ground_truth_dict": {
    "Bob": 1,
    "Peter": 2,
    "Alice": 3,
    "Arnold": 4,
    "Eric": 5,
    "dog": 1,
    "fish": 2,
    "bird": 3,
    "cat": 4,
    "horse": 5,
    "april": 1,
    "mar": 2,
    "jan": 3,
    "sept": 4,
    "feb": 5
},
"solve_dict_str": "{\n    \"Bob\": 5,\n    \"Eric\": 4,\n ...}",
"solve_response_time": 1.36,
"solve_token_usage": 585,
"solve_accuracy": 0.4666666666666667,
"solve_correct_fields": 7,
"solve_total_fields": 15,
"solve_parsed": {
```

```
    "Bob": 5,
    "Eric": 4,
    "Peter": 2,
    "Arnold": 3,
    "Alice": 1,
    "horse": 5,
    "dog": 1,
    "cat": 4,
    "fish": 3,
    "bird": 2,
    "sept": 5,
    "april": 1,
    "jan": 3,
    "mar": 2,
    "feb": 4
},
"convert_constraints": "{\n    \"houses_count\": 5,\n ...}",
"convert_response_time": 5.7,
"convert_token_usage": 1403,
"convert_solver_str": "N/A",
"convert_solver_accuracy": 0.0,
"convert_correct_fields": 0,
"convert_total_fields": 15,
"convert_solver_parsed": "N/A",
"official_z3_constraints_present": false,
"constraints_accuracy": 0.0,
"constraints_correct_fields": 0,
"constraints_total_fields": 1,
"error": "Z3 solver returned no solution for LLM constraints."
},
```


Listing 4.13: A successful run using the baseline strategy without explanations.

Design Notes

To ensure fault tolerance, the logger is built to handle malformed or partially missing outputs without terminating the evaluation run. If JSON parsing fails, a fallback message is logged and scoring is adjusted accordingly. This makes the logger resilient for large-scale batch runs across diverse LLM configurations and puzzle sets.

Final Remarks on logger

The logging framework plays a pivotal role in quantifying model performance across solving and constraint translation tasks. While direct constraint evaluation remains a secondary metric due to representational ambiguity, the system provides reliable, reproducible measures of task-level accuracy and performance metadata essential for comparative benchmarking.

4.7 Component Six: Benchmark Aggregator

The `benchmark.py` module is responsible for post-processing the results logged during each LLM run and aggregating performance metrics across different experiments. It provides both global and difficulty-bucketed summaries that help quantify how well models performed across solving and conversion tasks.

Functionality

This module reads from the cumulative `log.json` file and computes high-level statistics including:

- Average accuracy (exact matches).
- Weighted accuracy (cell-level correctness per puzzle).
- Total number of fields and completed runs.
- Token usage and response time aggregation.
- Stratified analysis by puzzle difficulty.

Each puzzle is grouped into one of four difficulty levelsSmall, Medium, Large, and X-Largebased on its size as per the classification scheme from ZebraLogic Lin et al. (2025). This categorization enables a nuanced understanding of model performance under varying complexity.

Metrics Captured

Three core evaluation domains are reported in the benchmark output:

- **Solve Accuracy:** Measures how often the LLM's direct dictionary output matched the ground truth, with both:
  - *Average Accuracy:* The proportion of runs where the solution was entirely correct.
  - *Weighted Accuracy:* The average ratio of correctly predicted fields per puzzle (cell-level accuracy).
- **Conversion Accuracy:** Applies the same metrics to the output derived from solving the puzzle via the Z3 solver using the LLMs generated constraints.
- **Constraint Accuracy (Optional):** When available, compares LLM-generated constraints to ground-truth formats. (As discussed in see *Constraint Format Matching (Limited)* Section Evaluation Scope). This metric is treated as exploratory due to its sensitivity to structural variations and constraint ambiguity. It is not included in final performance figures.

Output Structure

The benchmark script produces a `benchmarks_stats.json` file summarizing aggregated statistics. A simplified excerpt might look like:

```
{
  "solve": {
    "averages": {
      "average_accuracy": 0.62,
      "weighted_accuracy": 0.74,
      "total_fields": 700,
      "entries": 50
    }
  },
  "convert": {
    "averages": {
      "average_accuracy": 0.54,
      "weighted_accuracy": 0.72,
      "total_fields": 700,
```

```
      "entries": 50
    }
  },
  "constraints": {
    "averages": {
      "average_accuracy": 0.0,
      "weighted_accuracy": 0.0,
      "total_fields": 1,
      "entries": 0
    }
  },
  "by_difficulty": {
    "Small": {...},
    "Medium": {...},
    "Large": {...},
    "X-Large": {...}
  },
  "time_range": {
    "earliest": "2025-04-10_10:15:42",
    "latest": "2025-04-16_22:45:10"
  }
}
```

Listing 4.14: High-level statistics computed across log entries and grouped by difficulty.

Final Remarks on Benchmark

The benchmark module serves as a lightweight but effective tool for systematically assessing LLM behavior across multiple puzzle runs. It distills raw execution logs into interpretable summaries that support both model comparison and longitudinal analysis.

4.8 Summary

This chapter has detailed the practical realization of the experimental pipeline used to evaluate linguistic and functional competence in Large Language Models (LLMs) through Zebra puzzles. Each core component-ranging from the structured dataset and prompt generation strategies to the solver integration and logging infrastructurewas developed to support modular experimentation and reproducibility.

The `puzzle.py` file served as the foundation of the system, encoding the 50 puzzles used across all evaluations. The `main.py` controller coordinated interactions across modules, while `prompt_generator.py` enabled configurable prompting styles including baseline, Chain-of-Thought, and multi-shot. The `z3_solver.py` module translated JSON-formatted constraints into SMT representations and returned validated solutions. The `logger.py` module evaluated model outputs against ground truth and recorded detailed run metadata, and finally, `benchmark.py` aggregated and summarized performance statistics across dimensions such as accuracy, difficulty, and token usage.

By separating concerns and encapsulating logic within distinct modules, the system provides a flexible architecture for head-to-head evaluation of LLM behaviors across varied tasks. This modularity also supports future extensions-such as incorporating newer LLM variants, feedback-driven reasoning, or broader puzzle domains. The design and implementation together form a reproducible testbed for ongoing investigation into the reasoning capacities of modern language models.

Chapter 5

Evaluation

This chapter presents a comprehensive evaluation of the system developed for solving and translating Zebra-style logic puzzles using large language models (LLMs). The goal is to assess model performance across a wide spectrum of puzzle configurations, difficulty levels, and prompt strategies. Evaluation is carried out through automated benchmarking of solving accuracy, constraint conversion success, and token efficiency.

Five LLM variants were included in the study:

- **GPT-4o** and **GPT-o3-mini** from OpenAI
- **DeepSeek-Chat** and **DeepSeek-Reasoner**
- **Mistral-small-latest**

Each model was tested with three distinct prompting strategies:

- **Baseline** direct instruction with minimal context
- **Chain-of-Thought (CoT)** intermediate reasoning required before final output
- **Multishot** demonstrations of solved examples provided before the actual task

5.1 Experimental Setup

The evaluation uses a testbed of 50 Zebra puzzles sampled from four puzzle sizes: Small, Medium, Large, and X-Large. Each LLM + prompt strategy pair was run across the same subset of puzzles, with the following metrics logged:

- **Solve Accuracy:** Whether the full solution matched the ground truth

- **Solve Cell Accuracy:** Proportion of correctly assigned fields
- **Convert Accuracy:** Whether constraint-based solutions matched ground truth
- **Convert Cell Accuracy:** Per-field accuracy of solver outputs from LLM-generated constraints
- **Token Usage:** Total tokens used for both solve and convert prompts

Parameter	**Value**	**Description**
Puzzles Used	50 total	Ranging in size from 2x2 to 6x6
Models Tested	5 total	GPT-4o, GPT-o3-mini, DeepSeek-Chat, DeepSeek-Reasoner, Mistral
Prompt Strategies	3	Baseline, CoT, Multishot
Evaluation Modes	3	Solve, Convert, Full-Test
LLM Roles	`system`, `user`	Consistent instructions, enforced JSON outputs
Temperature	0.3	Low sampling temperature for stable outputs

Table 5.1: Overview of the controlled variables and evaluation settings.

5.2 Results

5.2.1 Overall Model Performance

This section presents an aggregate view of how each language model performed on the two primary tasks defined in this project: solving puzzles and converting textual clues into structured constraints. These results are averaged across all puzzle difficulties and prompt strategies to yield a global picture of each models general competence.

The top row of Figure 5.1 shows solve accuracy per model, including both strict solution accuracy (left) and more granular cell-level correctness (right). Full solution accuracy measures whether the model's entire output exactly matches the puzzle ground truth, whereas cell-level accuracy credits partially correct responses by evaluating each individual assignment.

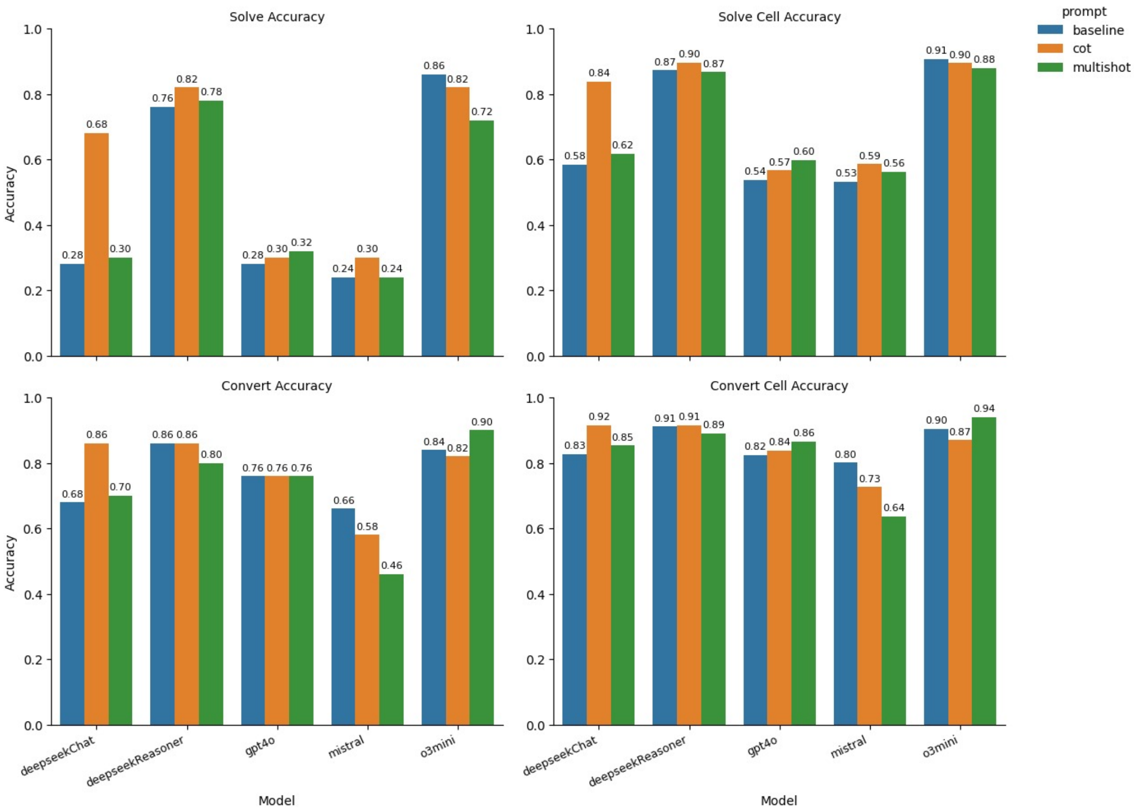

Figure 5.1: Comparison of model performance across both tasks. The top row shows solve accuracy: full puzzle correctness (left) and per-cell accuracy (right). The bottom row presents corresponding results for the convert task. Accuracy scores are shown for all models and prompt strategies.

Solve Task Performance

Among the models tested, **o3mini** and **DeepSeek Reasoner** consistently delivered the highest performance across both solve metrics. For example, o3mini achieved its best results of up to 0.86 full-puzzle accuracy and 0.91 cell-level accuracy under the baseline prompt strategy. However, its Reasoning compatriot did not lag far behind, demonstrating its best performance of 0.82 full-puzzle accuracy and 0.90 cell-level accuracy under the CoT prompt strategy.

In contrast, **GPT-4o**, **Mistral**, and **DeepSeek Chat** exhibited notably lower performance on full-puzzle accuracy, typically ranging from 0.24 to 0.32 across all prompt variants. Their cell-level accuracy remained moderate-between 0.53 and 0.62-indicating that while individual assignments were often correct, the models struggled to maintain global logical consistency. *(Notably, DeepSeek Chat demonstrated a significant performance boost under Chain-of-Thought prompting; this is explored further in Section 5.2.3.)*

Convert Task Performance

The bottom row of Figure 5.1 evaluates how well models translated natural language clues into structured constraint representations. As with solving, both full conversion accuracy and cell-level constraint correctness are reported.

o3mini emerged as the top performer, achieving up to 0.90 full conversion accuracy and 0.94 in cell-level constraint match under multishot prompting. **DeepSeek Reasoner** was similarly competitive, reaching 0.86 and 0.91 in the same respective metrics.

GPT-4o showed marked improvement in this task relative to solving, with full conversion accuracy stabilizing around 0.76 and cell-level performance peaking at 0.86.

DeepSeek Chat again delivered moderately strong results, especially Under Chain-of-Thought prompting, it achieved 0.86 in full-format accuracy and up to 0.92 in cell-level constraint correctness.

Mistral, however, continued to underperform, with full conversion accuracy peaking at just 0.66 and dropping to 0.46 in its weakest configurations. Cell-level accuracy remained low overall, reaching a maximum of only 0.80. Many of its outputs failed to parse due to malformed JSON or incomplete constraint schemas, highlighting limitations in handling structured reasoning formats.

Conversion vs Solving Performance: A Key Observation

Across nearly all evaluated configurations, models performed **better on the conversion task** than on solving-both in terms of full accuracy and per-cell correctness. This trend was especially pronounced in **non-reasoning models** such as **GPT-4o**, **Mistral**, and **DeepSeek Chat**, which consistently achieved higher accuracy on constraint translation compared to full logical solving.

This finding aligns with prior hypotheses proposed by Mahowald et al. Mahowald et al. (2024), who argue that models may demonstrate **stronger linguistic competence** (e.g., formatting, instruction following) than functional reasoning capacity. In contrast, **reasoning-specialized models** such as **o3mini** and **DeepSeek Reasoner** displayed smaller performance gaps between solving and converting, likely due to their ability to apply internal symbolic structure across both tasks.

This divergence supports the view that **conversion is more linguistically grounded**, whereas solving requires coherent multi-step reasoning. It further reinforces the need to evaluate both competencies jointlyusing tasks like Zebra puzzlesto meaningfully assess where LLMs excel or fall short.

Key Takeaways

- **Reasoning-oriented models**, particularly **o3mini** and **DeepSeek Reasoner**, consistently outperformed their general-purpose counterparts across both solve and convert tasks, achieving the highest scores in full and cell-level metrics.
- **Cell-level accuracy** served as a valuable diagnostic metric, highlighting latent capabilities even in underperforming models. For instance, GPT-4o and DeepSeek Chat showed moderate cell-level performance despite weak full-task accuracy.
- **Prompt strategies** had uneven effects: Chain-of-Thought prompting significantly enhanced DeepSeek Chats performance in both tasksmost notably in convertbut yielded only marginal improvements for GPT-4o and Mistral.
- **Structural fidelity and output formatting** emerged as core challenges for weaker models. Mistral, in particular, suffered from frequent formatting and schema issues that degraded performance, especially in the constraint translation task.

5.2.2 Robustness Across Puzzle Difficulties

This section evaluates the extent to which model performance degrades as the puzzle complexity increases. Puzzle instances in the dataset range from Small (2x2) to Extra-Large (6x6), and this analysis investigates whether models are capable of maintaining reasoning fidelity as combinatorial complexity grows.

Analysis and Observations

Figures 5.2 and 5.3 illustrate how model accuracy scales with puzzle difficulty. As expected, most models exhibited a decline in performance as puzzle size increased, with X-Large instances presenting the greatest challenge.

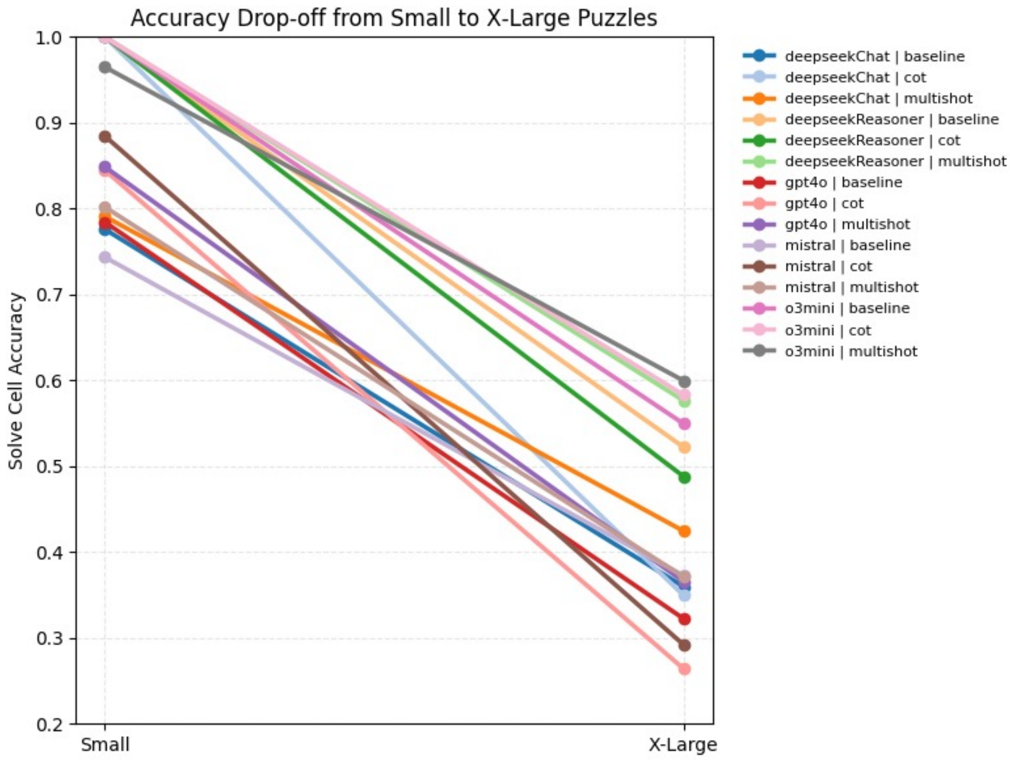

Figure 5.2: Solve cell accuracy for each model and prompt configuration, plotted from Small to X-Large puzzles. The drop in performance reflects how well models scale with increasing reasoning complexity.

The accuracy drop-off plot (Figure 5.2) shows that **o3mini** and **DeepSeek Reasoner** maintained the highest accuracy across the spectrum, although both still experienced a significant decline between Small and X-Large puzzles. For instance, o3minis solve cell accuracy dropped from nearly 1.00 on Small puzzles to 0.55 on X-Large ones.

Conversely, **GPT-4o**, **Mistral**, and **DeepSeek Chat** were more severely impacted by increased puzzle complexity, with several configurations dropping below 0.30 accuracy on X-Large tasks.

The heatmap in Figure 5.3 provides a more granular breakdown by combining model and prompt variant on the horizontal axis, and puzzle difficulty on the vertical axis. It highlights how some prompting strategies (notably Chain-of-Thought for DeepSeek Chat and DeepSeek Reasoner) substantially mitigate performance loss on Medium and Large puzzles, though not fully eliminating it.

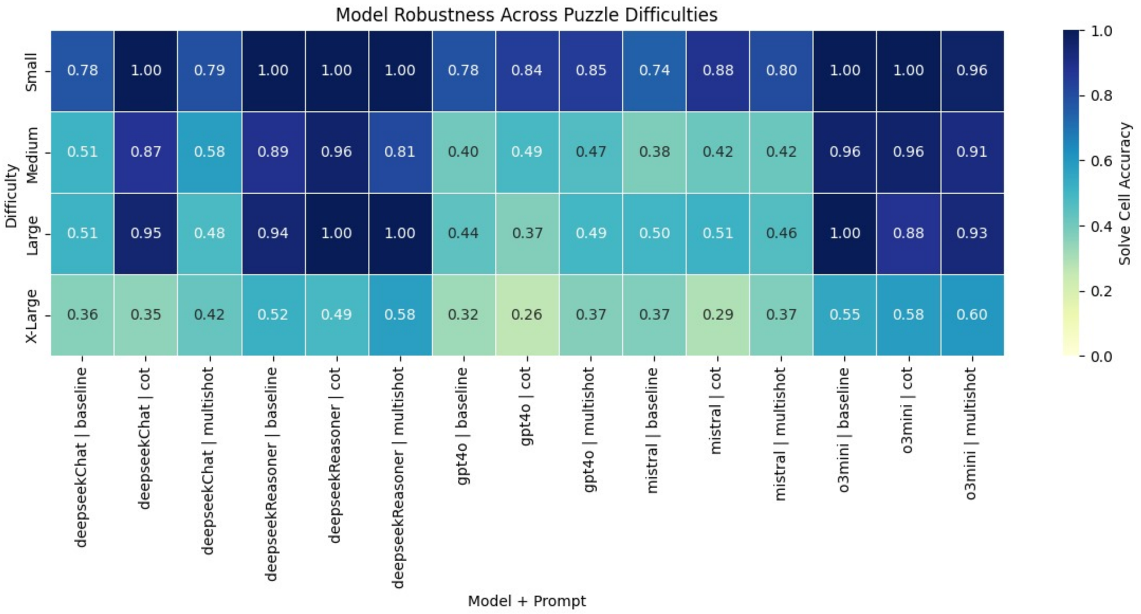

Figure 5.3: Cell-level accuracy across puzzle difficulties for each model-prompt pair. Rows correspond to puzzle sizes, and columns correspond to prompt strategies per model.

Key Observations

- **All models degrade** with increasing puzzle size, but reasoning-focused models exhibit slower performance decay.

- **Prompt strategies** help counterbalance difficulty effectsparticularly for DeepSeek models under CoT and multishot prompting. General-purpose models like **GPT-4o** and **Mistral** exhibit substantial degradation on X-Large puzzles, with solve cell accuracy dropping below 0.32. This suggests limited capacity to scale their reasoning to handle long-range constraints and multi-step dependencies.

5.2.3 Prompt Strategy Impact

This section evaluates the effect of different prompt strategies-*Baseline*, *Chain-of-Thought (CoT)*, and *Multishot*-on the performance of each model. Rather than focusing on absolute accuracy, Figures 5.4 and 5.5 present the **lift over baseline**, i.e., the change in cell-level accuracy attributable to advanced prompting techniques.

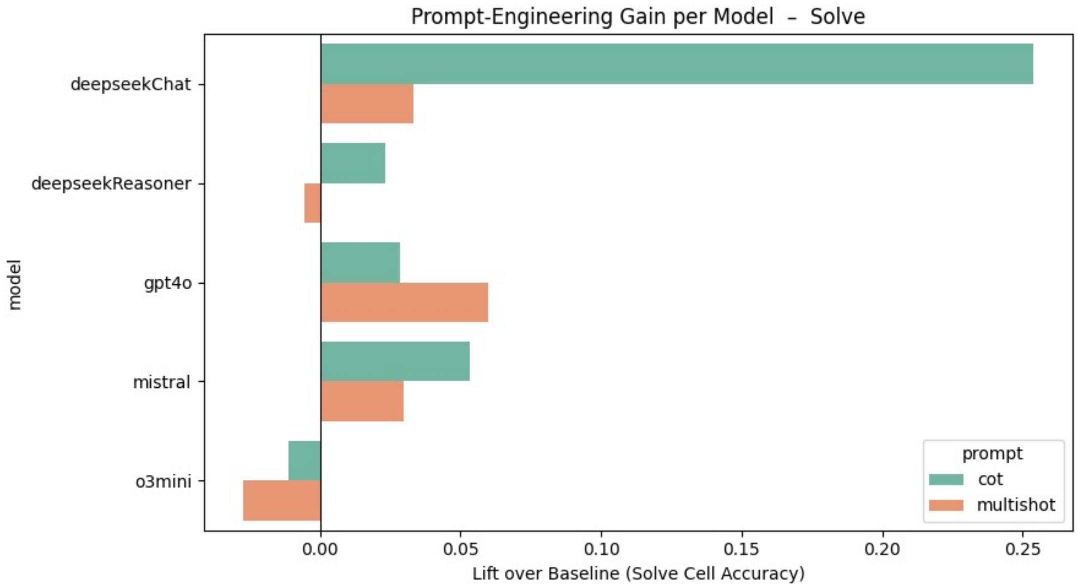

Figure 5.4: Change in solve cell-level accuracy compared to baseline prompt for each model.

Prompt Effectiveness on the Solve Task

Figure 5.4 illustrates the gain in solve cell accuracy for each model when using Chain-of-Thought (CoT) or multishot prompting, relative to its baseline configuration.

The most pronounced gains are observed in **DeepSeek Chat**, which improves by over **0.25** when using CoT prompts. This dramatic lift suggests that external reasoning scaffolds-such as explicitly walking through logical deductions-can significantly enhance performance in models that are not explicitly optimized for multi-step reasoning but possess the contextual capacity and model size to benefit from such structured guidance.

Other general-purpose models such as **GPT-4o** and **Mistral** also benefited modestly from structured prompting, with Chain-of-Thought yielding gains of 0.02 and 0.05 respectively, and multishot prompting yielding slightly higher improvements of 0.06 and 0.03. While these gains are relatively modest, they demonstrate that external scaffolding can guide models with limited inherent reasoning ability toward more consistent outputs, even if the underlying logical capabilities remain unchanged. Notably, the effectiveness of a given prompt strategy is not uniform across models-some models respond more positively to Chain-of-Thought, while others show greater improvement under multishotsuggesting that optimal prompting may be architecture-specific rather than universally transferable.

In contrast, the **reasoning-specialized models**, **o3mini** and **DeepSeek Reasoner**, exhibit minimal or even negative changes relative to their baseline prompts. For o3mini in particular, some prompt variants led to small performance regressions. This suggests that such models already operate with strong internal logical priors, and further prompting may either be redundant or introduce unnecessary verbosity that dilutes their reasoning focus.

This pattern reveals a broader insight: **non-reasoning models consistently benefit from prompt engineering**, while **reasoning-capable models are less sensitive-or even slightly averse to it**. Prompt scaffolding appears to serve as a crucial external aid when internal reasoning structures are lacking, but may interfere with models that are already optimized for reasoning.

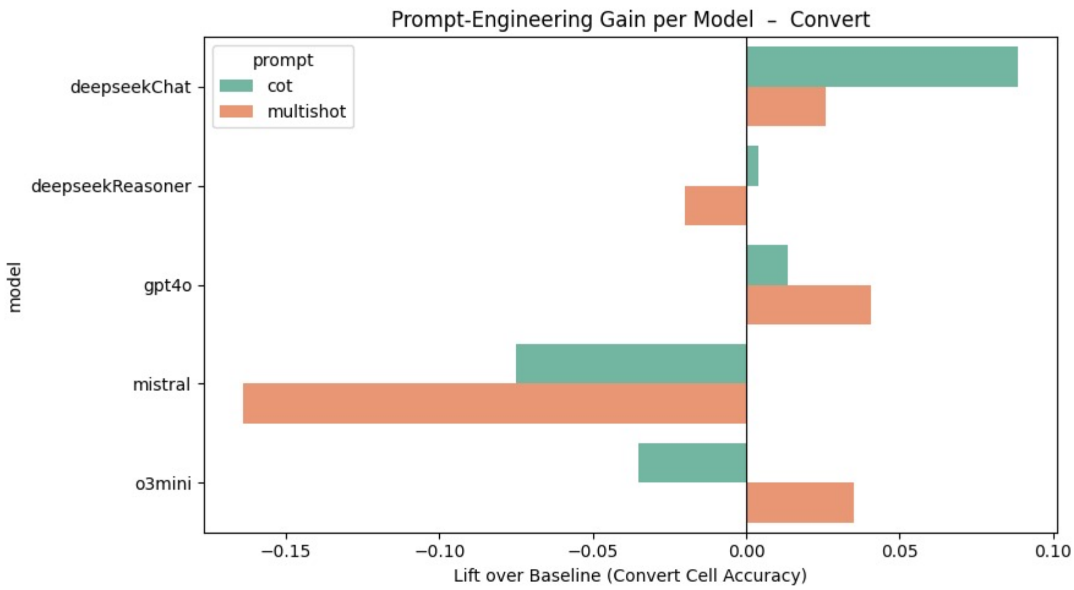

Figure 5.5: Change in convert cell-level accuracy compared to baseline prompt for each model.

Prompt Effectiveness on the Convert Task

Figure 5.5 presents the relative improvement in cell-level accuracy for the convert task under different prompting strategies. Unlike the solve task, where prompting had a more substantial impact on weaker models, gains here were more modest overall. This is likely due to the lower reasoning requirements of constraint translation, which prioritizes structural precision over multi-step inference.

As in the solve task, **DeepSeek Chat** exhibited the largest improvement, gaining nearly **0.10** in cell-level accuracy under Chain-of-Thought prompting. This highlights the role of prompt clarity in helping verbose or instruction-sensitive models output well-structured constraints.

By contrast, **Mistral** demonstrated severe sensitivity to prompt complexity. Under CoT and multishot prompting, it lost up to 15 percentage points in accuracy, often producing malformed or incomplete JSONhighlighting challenges in structured text generation under verbosity.

o3mini and **DeepSeek Reasoner** again remained relatively unaffected by prompt variation, maintaining consistently high performance across all strategies. This prompt-invariance suggests that their formatting and reasoning capabilities are internally robust and less reliant on scaffolding. **GPT-4o** showed minor improvements under multishot, suggesting some responsiveness to example-based prompting.

Across both tasks and all models, it became evident that the optimal prompt strategy varied by model architecture and capability. While my initial hypothesis assumed that Chain-of-Thought would be best suited for solving (by enhancing reasoning) and multishot for conversion (by improving formatting fidelity), the results show that Chain-of-Thought generally outperformed others in the solve task, while multishots impact on conversion was inconsistent and model-dependent. This suggests that prompt effectiveness is more nuanced than originally expected and may interact in complex ways with model design, size, and training emphasis.

Key Observations

- **DeepSeek Chat** benefits most reliably from Chain-of-Thought prompting, especially in converting natural language to structured formats.
- **Mistral** is highly susceptible to prompt verbosity, with CoT and multishot strategies often degrading performance in the convert task.
- **o3mini** and **DeepSeek Reasoner** maintain strong performance regardless of prompting style, indicating robustness to prompt variation.
- **GPT-4o** saw slight benefits from multishot prompting, but its performance remained moderate overall.

- **Across both tasks**, Chain-of-Thought prompting emerged as the most generally beneficial strategy, especially for reasoning, whereas multishot prompting produced variable results and did not consistently outperform the baseline.

- **Prompt effectiveness is model-dependent**, with no universally optimal strategyhighlighting the importance of tuning prompt style based on model architecture and task complexity.

5.2.4 Reasoning-Oriented vs General-Purpose Models

To better understand the impact of architectural and training design on puzzle-solving capabilities, we grouped models into two categories: **reasoning-oriented** models (o3mini, DeepSeek Reasoner) and **general-purpose** models (GPT-4o, Mistral, DeepSeek Chat). These two cohorts differ in their optimization priorities-while reasoning models are explicitly trained to handle symbolic and multi-step logic, general-purpose LLMs emphasize speed, versatility, or open-ended generation.

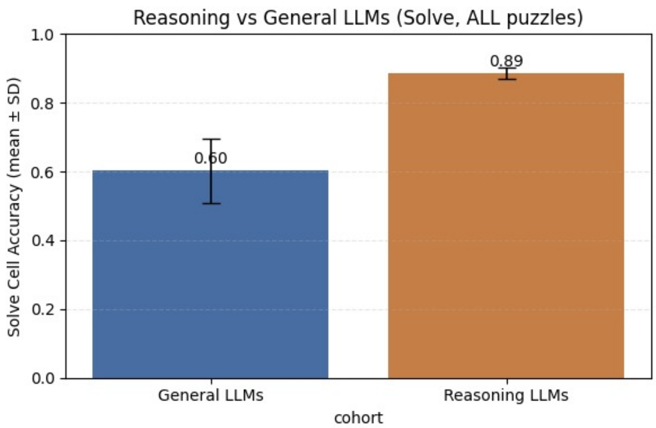

Figure 5.6: Average solve cell accuracy across all puzzles and prompt-strategies for reasoning-oriented and general-purpose LLMs. Error bars denote standard deviation.

As shown in Figures 5.6 and 5.7, reasoning-oriented models substantially outperform their general-purpose counterparts. On the **solve task**, the average cell-level accuracy for reasoning models reaches **0.89**, compared to just **0.60** for general models. This highlights a considerable performance gap when tasks require step-wise logical deduction and structural consistency.

For the **convert task**, while the difference is less pronounced, it remains significant:

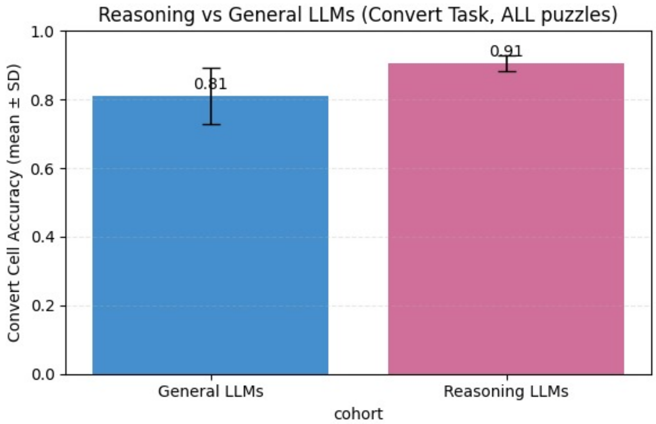

Figure 5.7: Average convert cell accuracy across all puzzles and prompt-strategies for reasoning-oriented and general-purpose LLMs. Error bars denote standard deviation.

reasoning models achieve an average accuracy of **0.91** compared to **0.81** for general models. This narrower margin suggests that convert tasksbeing more syntactic and less dependent on deep logical chaining-can be handled reasonably well by models with sufficient formatting capabilities.

These results reinforce the importance of architectural specialization. Despite having similar scale or context capacity, general-purpose LLMs struggle to match the precision and consistency of reasoning-optimized models, particularly in complex structured tasks like solving logical puzzles.

5.2.5 Token Efficiency: Accuracy vs Token Budget

Beyond accuracy, practical deployment of language modelsespecially in interactive or API-constrained environmentsdemands careful consideration of computational efficiency. Token usage directly impacts latency, cost, and throughput. This section evaluates token efficiency for both tasks by plotting accuracy against total token usage (log-scaled).

In the solve task (Figure 5.8), reasoning-optimized models like **o3mini** and **DeepSeek Reasoner** dominate the upper-right quadrant-offering strong accuracy but at a higher token cost. Despite their verbosity, the performance boost justifies the expense in settings where correctness is paramount. Notably, **DeepSeek Chat with Chain-of-Thought prompting** achieves solve accuracy nearing 0.86 while consuming significantly fewer tokens, suggesting that well-structured prompts can unlock latent reasoning capabilities in general-purpose models. As seen in the overall accuracy chart (Figure 5.1), this particular

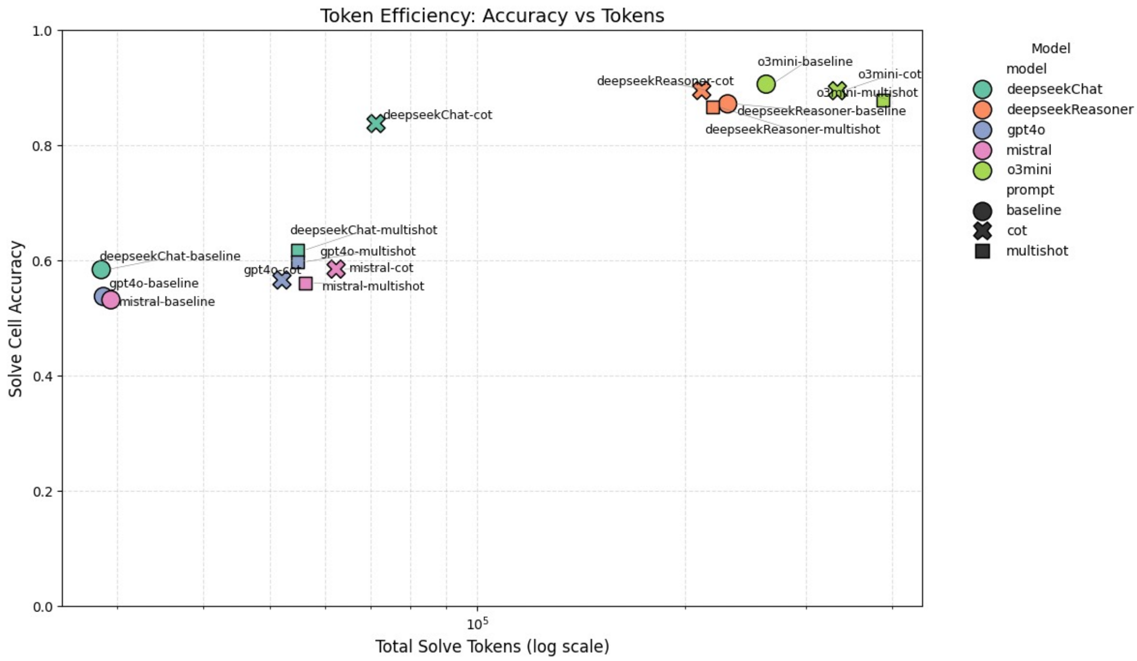

Figure 5.8: Solve cell accuracy versus total tokens consumed for each model + prompt combination. High-performing configurations appear toward the top-left (high accuracy, low tokens).

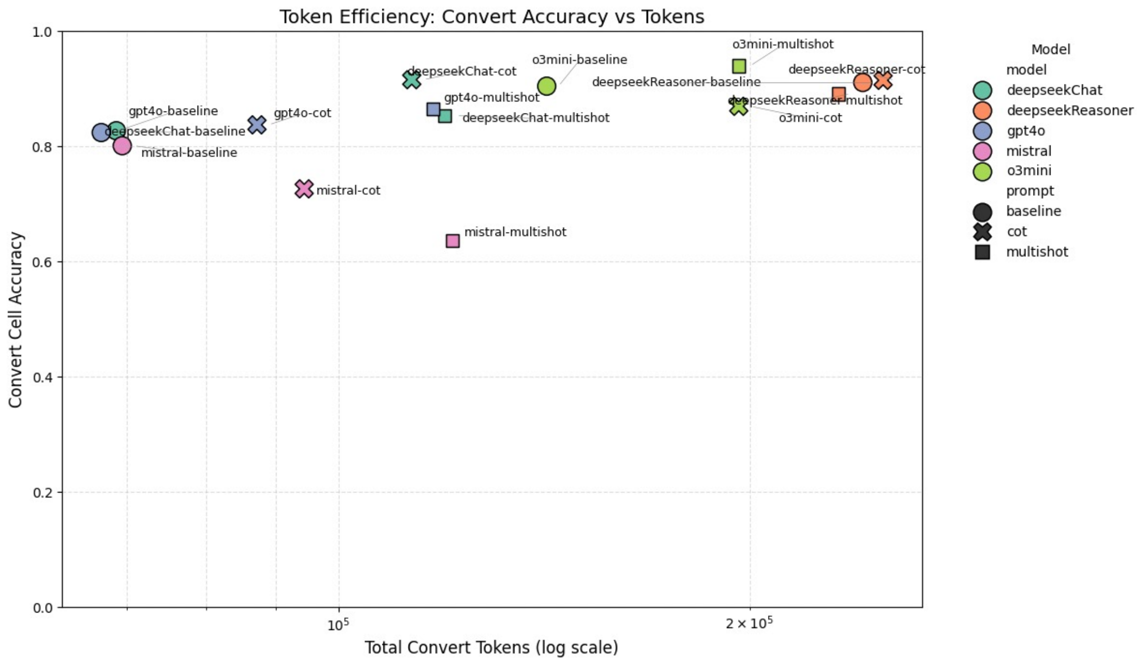

Figure 5.9: Convert cell accuracy versus token usage across prompt-model pairs.

configuration outperforms several reasoning LLMs, indicating that with the right prompting, non-reasoning models can be a superior choice in constrained environments.

In contrast, general models like **Mistral** and **GPT-4o** show weaker solve performance despite being more token-efficient, making them unsuitable for solving tasks when accuracy is the primary objective.

The convert task (Figure 5.9) presents a different picture. Accuracy levels across models are more tightly clustered-often within 10 points. In this scenario, models such as **GPT-4o** and **DeepSeek Chat** provide a compelling trade-off: they deliver cell-level accuracy of 0.860.88 at a much lower token cost compared to reasoning models. Given the reduced need for complex inference in this task, this supports the case for using non-reasoning models in practical pipelines focused on clue translation and constraint generation.

Mistral once again appears the least efficient, suffering from both lower accuracy and increased verbosity under multishot or CoT prompting.

Note on DeepSeek Chat Performance

While **DeepSeek Chat** with Chain-of-Thought prompting exhibits excellent token efficiency, it also suffers from a steep performance drop as puzzle complexity increases-particularly on Large and X-Large puzzles (see Figure 5.2). In contrast, reasoning-specialized models such as **o3mini** and **DeepSeek Reasoner** maintain more stable accuracy across difficulty tiers. This suggests that while DeepSeek Chat is an efficient option for simpler tasks, its performance does not scale well to complex multi-step reasoning problems, where robustness is better supported by dedicated architectures.

Token Efficiency Insights

- **o3mini**, although highly accurate, incurs a significant token cost. Its use may be justified in contexts where peak performance is critical and cost is secondary.
- **DeepSeek Chat (CoT)** delivers one of the best token-to-accuracy tradeoffs in the solve task-even outperforming reasoning LLMs under the right prompt structure (see Figure 5.1).
- For the **convert task**, accuracy differences are relatively small across models, making general-purpose models such as **GPT-4o** and **DeepSeek Chat** more attractive

from a cost-efficiency perspective.

- **Prompt strategy** strongly influences token use: multishot prompts consistently incur higher costs and do not always result in better performance.

Chapter Summary

This chapter systematically evaluated the performance of several large language models across structured reasoning tasks, focusing on both full-puzzle solving and constraint conversion. Results were analyzed using a wide variety of metrics, including full and cell-level accuracy, robustness across difficulty levels, sensitivity to prompt strategies, and token efficiency.

Key findings include:

- **Reasoning-specialized models** such as **o3mini** and **DeepSeek Reasoner** consistently outperformed general-purpose models in both tasks, particularly in solving where logical coherence and constraint propagation were essential.

- **Prompt engineering** was shown to substantially improve performance in weaker models. Chain-of-Thought prompting notably elevated DeepSeek Chats solve accuracy by over 0.25, closing the gap with more advanced LLMs. However, its impact varied significantly across models and tasks.

- **Robustness to puzzle difficulty** was a key differentiator. Most models exhibited a sharp decline in performance on X-Large puzzles, with only reasoning-focused LLMs maintaining relative stability. This underscores the importance of model architecture and inductive bias in scaling symbolic reasoning.

- **Token efficiency** analyses revealed trade-offs between performance and cost. While **o3mini** achieved the highest accuracy, it was the most token-intensive. In contrast, **DeepSeek Chat** with CoT prompting demonstrated strong performance with relatively low token usage suggesting it may be a more cost-effective choice in constrained deployments. However, these gains were not sustained at higher difficulty levels, where DeepSeek Chat's accuracy dropped sharply, highlighting the limitations of efficiency without architectural reasoning depth.

- **Convert tasks** exhibited narrower performance gaps between model types, indicating that such tasks demand less global reasoning and benefit more uniformly from good formatting and prompt clarity. This raises interesting implications for real-world pipeline design, where cheaper LLMs may suffice for pre-processing or data transformation steps.

- **Critically, models-especially non-reasoning ones-were consistently better at the convert task than solving**, confirming the hypothesis that linguistic competence (e.g., formatting, instruction adherence) remains stronger than logical inference in most LLMs. Reasoning-specialized models showed less of a gap, indicating more unified capabilities.

Overall, these results demonstrate the value of combining architectural capabilities with careful prompt design. They also highlight the need for differentiated strategies when choosing models for high-reasoning tasks (e.g., solving Zebra puzzles) versus structurally simpler ones (e.g., constraint translation). The next chapter synthesizes these insights into actionable conclusions and suggests directions for future work.

Chapter 6

Conclusions & Future Work

This chapter revisits the central goals of the dissertation and summarizes the conclusions drawn from the experimental results and analysis presented in Chapter 5. It also outlines potential directions for future research, based on both observed limitations and promising opportunities that emerged throughout the project.

6.1 Summary of Findings

This dissertation set out to investigate the comparative linguistic and functional competence of modern large language models (LLMs), using Zebra puzzles as a structured, interpretable testbed. The project was guided by three core research goals as outlined in Section 1.1:

1. **Comparative Competence:** Determine whether LLMs perform better on linguistic tasks (e.g., constraint translation) than on functional tasks (e.g., direct logical reasoning), as hypothesized by Mahowald et al. Mahowald et al. (2024).
2. **Prompt Engineering Impact:** Examine whether prompt strategies such as Chain-of-Thought and multishot prompting measurably improve model accuracy on both linguistic and functional tasks.
3. **Benchmarking Current Capabilities:** Provide a representative snapshot of the reasoning ability of current state-of-the-art LLMs on structured logical problems.

1. Comparative Competence. The results strongly support the initial hypothesis: across models and prompt types, LLMs were more effective at converting natural language clues into structured formats than at solving the puzzles directly. This trend was especially pronounced in non-reasoning models like **GPT-4o**, **DeepSeek Chat**, and **Mistral**. However, the performance gap narrowed considerably in reasoning-specialized models such as **o3mini** and **DeepSeek Reasoner**, whose solving and conversion accuracy were comparably high. These findings are discussed in detail in Section 5.2.1.

2. Prompt Engineering Impact. Prompting strategies had a meaningful impact, particularly for general-purpose models. Chain-of-Thought prompting significantly boosted performance on the solve task, especially for DeepSeek Chat, which improved by over 0.25 in accuracy. Multishot prompting yielded more mixed results-effective in some cases (notably o3mini's convert task) but detrimental in others (e.g., Mistral's convert performance). Reasoning models were generally less sensitive to prompting strategy, suggesting that their capabilities are more robust and less reliant on scaffolding. Detailed prompt-specific insights are presented in Section 5.2.3.

3. Benchmarking Current Capabilities. The study offers a clear empirical benchmark of LLM reasoning as of 2025. **o3mini** and **DeepSeek Reasoner** consistently outperformed general models across all tasks, demonstrating strong symbolic reasoning and format adherence. Meanwhile, models like **DeepSeek Chat** exhibited competitive accuracy at a fraction of the token cost, indicating viable deployment scenarios in constrained environmentsespecially for the convert task. This trade-off between accuracy and efficiency is further examined in Section 5.2.5.

Additional Observations.

- **Cell-level metrics** revealed partial correctness in otherwise incorrect outputs, particularly for GPT-4o and Mistral-making them an essential tool in nuanced LLM evaluation.

- **Prompt performance varied by model and task.** Chain-of-Thought generally outperformed others in the solve task, but no universal strategy emerged for convert. Gains were inconsistent across models, and in some cases (e.g., Mistral), verbose prompts led to degraded outputs.

- **DeepSeek Chat (CoT)** offered excellent token efficiency and strong conversion accuracy, but suffered a steep performance drop on large and extra-large puzzles-unlike reasoning models that maintained robustness across puzzle sizes (see Figure 5.3).

- **Task type matters.** While solving required multi-step deduction and structure tracking, convert tasks often benefitted more from linguistic and syntactic adherence. This difference may explain why prompt strategies had a greater impact on solve than convert.

6.2 Future Work

Based on the outcomes and limitations of this study, several avenues for future research are proposed:

- **Automated Constraint Repair:** A significant portion of conversion failures stemmed from minor syntactic or logical errors. Future systems could address this by implementing a feedback loop between the LLM and the Z3 solver, where failed outputs are iteratively revised based on solver error messages until a satisfiable solution is reached. This approach mirrors techniques proposed by Berman et al. Berman et al. (2024), who employed constraint-guided correction mechanisms to successfully solve Zebra puzzles via iterative agent collaboration.

- **Reasoning Chain Evaluation:** While Chain-of-Thought prompts encouraged step-wise reasoning, the quality of intermediate steps was not assessed in this dissertation. Future work could annotate and score these explanations to better understand model transparency and truthfulness.

- **Beyond Zebra Puzzles:** Although Zebra puzzles provided a clean and structured reasoning framework, extending this analysis to other formal domains could offer broader insights.

- **Larger Puzzle Dataset:** The current evaluation was based on 50 puzzles of varying complexity. Expanding this benchmark to include hundreds of puzzles-especially those with more complex or ambiguous clues-would improve statistical robustness and stress-test model generalization.

- **Constraint-Level Evaluation:** The current analysis of constraint translation accuracy was limited to the first ten puzzles due to time constraints and the manual effort required to produce ground truth constraint annotations. Completing this annotation effort would enable a more granular and reliable assessment of conversion performance, particularly across prompt types and puzzle sizes. See 3.1.1

- **Incorporating New LLMs:** Several newer models-such as GPT-4.5, OpenAIs O1-Pro, Claude 3, and Metas LLaMA 3-have shown promise in early benchmarks but were excluded from this study due to financial and access constraints. Future work should include these models to reassess the relative performance gap between general and reasoning-specialized LLMs.

- **Exploring Alternative Prompt Strategies:** This study focused on Zeroshot(baseline), Chain-of-Thought, and multishot prompting. Future research could examine additional prompt engineering techniques such as Tree of Thought(ToT), Prompt Chaining or instruction tuning to further enhance model performanceparticularly for general-purpose LLMs.

- **Simplifying the Conversion Template:** The current constraint conversion schema requires adherence to a relatively strict, complex, and verbose JSON format. Restructuring the schema to be simplerby reducing the number of allowable constraint types, or adopting a more compact syntaxcould mitigate failure modes related to formatting inconsistencies. This could be especially beneficial for weaker models that tend to hallucinate or misinterpret specification rules, and may also streamline downstream parsing and solver integration.

In conclusion, while state-of-the-art LLMs have made significant strides in symbolic reasoning and structured representation, achieving stable and high-accuracy performance across all reasoning tasks and puzzle complexities remains an open challenge. This study contributes a clear snapshot of the current landscape-serving both as a benchmark for evaluating progress and a blueprint for advancing LLM reasoning capabilities through more refined architectures, prompt designs, and evaluation methodologies.

Bibliography

Anthropic (2025). Claude 3.7 sonnet and claude code. `https://www.anthropic.com/news/claude-3-7-sonnet`.

Berman, S., McKeown, K., and Ray, B. (2024). Solving zebra puzzles using constraint-guided multi-agent systems. *arXiv preprint arXiv:2407.03956*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

DataCamp (2024). Chain-of-thought prompting. `https://www.datacamp.com/tutorial/chain-of-thought-prompting`.

De Moura, L. and Bjrner, N. (2008). Z3: An efficient smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS, pages 337–340. Springer.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

Fourrier, C., Habib, N., Lozovskaya, A., Szafer, K., and Wolf, T. (2024). Open llm leaderboard v2. `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`.

Gauthier, J., Hu, J., Wilcox, E., Qian, P., and Levy, R. (2020). Syntaxgym: An online platform for targeted evaluation of language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 70–76. Association for Computational Linguistics.

Giadikiaroglou, P., Lymperaiou, M., Filandrianos, G., and Stamou, G. (2024). Puzzle solving using reasoning of large language models: A survey. *arXiv preprint arXiv:2402.11291*.

Groza, A. (2023). Can chatgpt solve the zebra puzzle? *arXiv preprint arXiv:2303.15047*.

Lin, B. Y., Bras, R. L., Richardson, K., Sabharwal, A., Poovendran, R., Clark, P., and Choi, Y. (2025). Zebralogic: On the scaling limits of llms for logical reasoning. *arXiv preprint arXiv:2310.05993*.

Mahowald, K., Ivanova, A. A., Blank, I. A., Kanwisher, N., Tenenbaum, J. B., and Fedorenko, E. (2024). Dissociating language and thought in large language models. *Trends in Cognitive Sciences*.

OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI (2025). Reasoning models. https://platform.openai.com/docs/guides/reasoning?api-mode=responses.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, A., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35.

Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. In *Computational Intelligence*. Springer.

Rae, J. W., Borgeaud, S., Cai, T., et al. (2021). Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.

Talwar, A. (2024). Multi-shot prompting explained simply. https://medium.com/@anmoltalwar/multi-shot-prompting-15a7c4b8b78e. Accessed: April 2025.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

Warstadt, A., Parrish, A., Liu, H., Mohananey, A., Peng, S.-F., Wang, P. M., and Bowman, S. R. (2020). Blimp: The benchmark of linguistic minimal pairs for english. *Transactions of the Association for Computational Linguistics*, 8:377–392.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Wikipedia contributors (2025). Zebra puzzle — Wikipedia, the free encyclopedia. [Online; accessed 15-April-2025].

Xu, H. et al. (2023). What do you know before the prompt? measuring inductive biases in pre-trained models. *arXiv preprint arXiv:2305.14745*.

Z3TheoremProver (2012). Microsoft research. https://www.microsoft.com/en-us/research/project/z3-3/.

Zhou, J., Lu, T., Mishra, S., Brahma, S., Basu, S., Luan, Y., Zhou, D., and Hou, L. (2023). Instruction-following evaluation for large language models.

Appendix A

Title of AppendixA

A.1 Project Repository

The full source code, configuration files, prompt templates, logs, and evaluation scripts used in this dissertation are available at:

`https://github.com/soodt/LogicalReasoning_LLMs`

A.2 Puzzle Format Examples

Puzzle with Structured Constraints


```
{
    "puzzle_1": {
        "text_description": "There are five houses. The
            Englishman lives in the red house. The Spaniard owns
            the dog. Coffee is drunk in the green house. The
            Ukrainian drinks tea. The green house is immediately
            to the right of the ivory house. The Old Gold smoker
            owns snails. Kools are smoked in the yellow house.
            Milk is drunk in the middle house. The Norwegian
            lives in the first house. The man who smokes
            Chesterfields lives in the house next to the man with
             the fox. Kools are smoked in the house next to the
            house where the horse is kept. The Lucky Strike
            smoker drinks orange juice. The Japanese smokes
            Parliaments. The Norwegian lives next to the blue
```

```
    house. Now, who drinks water? Who owns the zebra?",
  "size": "5x5",
  "z3_format": {
      "houses_count": 5,
      "categories": {
          "nationalities": ["Englishman", "Spaniard", "
              Ukrainian", "Norwegian", "Japanese"],
          "colors": ["Red", "Green", "Ivory", "Blue", "
              Yellow"],
          "pets": ["Fox", "Horse", "Zebra", "Dog", "Snails
              "],
          "drinks": ["Coffee", "Milk", "OrangeJuice", "Tea
              ", "Water"],
          "cigarettes": ["Parliaments", "Kools", "
              LuckyStrike", "OldGold", "Chesterfields"]
      },
      "constraints": [
          {
              "type": "distinct_categories",
              "categories": [
                  "nationalities",
                  "colors",
                  "pets",
                  "drinks",
                  "cigarettes"
              ]
          },
          {
              "type": "range",
              "from": 1,
              "to": 5
          },
          {"type": "eq", "var1": "Englishman", "var2": "
              Red"},
          {"type": "eq", "var1": "Spaniard", "var2": "Dog"
              },
```

```
            {"type": "eq", "var1": "Coffee", "var2": "Green"
              },
            {"type": "eq", "var1": "Ukrainian", "var2": "Tea
              "},
            {"type": "eq_offset", "var1": "Green", "var2": "
              Ivory", "offset": 1},
            {"type": "eq", "var1": "OldGold", "var2": "
              Snails"},
            {"type": "eq", "var1": "Kools", "var2": "Yellow"
              },
            {"type": "eq", "var1": "Milk", "var2int": 3},
            {"type": "eq", "var1": "Norwegian", "var2int":
              1},
            {"type": "neighbor", "var1": "Chesterfields", "
              var2": "Fox"},
            {"type": "neighbor", "var1": "Kools", "var2": "
              Horse"},
            {"type": "eq", "var1": "LuckyStrike", "var2": "
              OrangeJuice"},
            {"type": "eq", "var1": "Japanese", "var2": "
              Parliaments"},
            {"type": "neighbor", "var1": "Norwegian", "var2"
              : "Blue"}
        ]
    },
    "ground_truth_dict": {
        "Englishman": 3,
        "Spaniard": 4,
        "Ukrainian": 2,
        "Norwegian": 1,
        "Japanese": 5,
        "Red": 3,
        "Green": 5,
        "Ivory": 4,
        "Blue": 2,
        "Yellow": 1,
        "Fox": 1,
```

```
            "Horse": 2,
            "Zebra": 5,
            "Dog": 4,
            "Snails": 3,
            "Coffee": 5,
            "Milk": 3,
            "OrangeJuice": 4,
            "Tea": 2,
            "Water": 1,
            "Parliaments": 5,
            "Kools": 1,
            "LuckyStrike": 4,
            "OldGold": 3,
            "Chesterfields": 2
        }
    },
    ...
    "puzzle_50": {
        "text_description": "There are 4 houses, numbered 1 to 4
            from left to right, as seen from across the street.
            Each house is occupied by a different person. Each
            house has a unique attribute for each of the
            following characteristics: — Each person has a unique
            name: `Alice`, `Arnold`, `Eric`, `Peter` — Each
            mother is accompanied by their child: `Samantha`, `
            Bella`, `Meredith`, `Fred` — People have unique hair
            colors: `red`, `black`, `brown`, `blonde`  ## Clues:
            1. The person who has red hair is directly left of
            Eric. 2. The person whose child is named Meredith is
            directly left of the person who has brown hair. 3.
            The person who has blonde hair is the person whose
            child is named Fred. 4. Peter is in the fourth house.
             5. The person who has black hair is directly left of
             Arnold. 6. The person whose child is named Bella is
            Arnold.",
        "size": "4x3",
        "ground_truth_dict": {
```

```
            "Alice": 1,
            "Eric": 2,
            "Arnold": 3,
            "Peter": 4,
            "Samantha": 1,
            "Meredith": 2,
            "Bella": 3,
            "Fred": 4,
            "red": 1,
            "black": 2,
            "brown": 3,
            "blonde": 4
        }
    }
}
```

A.3 Prompt Strategies

```
{
CoT- Solve:
Think critically and show your step-by-step reasoning in the "
   explanation" field to solve the puzzle below. Then produce a
   final JSON with exactly these two fields:
{
"explanation": "...",
"solution": { ... }
}
**Do not** use triple-backticks or any code fences.
**Do not** include any extra text outside the JSON.
**Do not** add Markdown headings or bullet points.
Your entire response must be valid JSON, period.

...(Same as Baseline - Solve)...

Therefore, final output must be valid JSON in the form:
{
```

```
"explanation": "<chain-of-thought reasoning>",
"solution": {
    "Red": 1,
    "Blue": 2,
    "PersonA": 1,
    "PersonB": 2
    ...
}

CoT - Convert:

Think critically and, show your step-by-step chain-of-thought
   reasoning (explanation).
Then, produce a final JSON with two keys: "explanation" for your
    reasoning, and "z3" for the actual constraints.
Below are the rules for converting puzzles to Z3:

...(Same as Baseline Convert)...

Therefore, final output must be valid JSON in the form:
{
"explanation": "<chain-of-thought reasoning>",
"z3": {
"houses_count": 5,
"categories": {
    "colors": ["Red","Green","Blue","Yellow","White"],
    "people": ["Alice","Bob","Charlie","Diana","Evan"]
},
"constraints": [
    {
    "type":"distinct_categories",
    "categories":["colors","people"]
    },
    {
    "type":"range",
    "from":1,
    "to":5
```

```
    },
    { "type":"eq","var1":"Alice","var2":"Red" },
    { "type":"eq_offset","var1":"Blue","var2":"Green","offset":1
        },
    { "type":"neighbor","var1":"Bob","var2":"Alice" }

"""
Multishot - Solve:

... (Same as Basekine Solve) ...

For this question: There are two houses. One is red, the other
   is blue. Person A lives in the red house, and Person B lives
   in the blue house. Who lives in each house?
The solution would be:
{
"Red": 1,
"Blue": 2,
"PersonA": 1,
"PersonB": 2
}

For this question: here are 3 houses, numbered 1 to 3 from left
   to right. Each house is occupied by a different person. Each
   house has a unique attribute for each characteristic: - Names
   : Peter, Eric, Arnold - Drinks: tea, water, milk ## Clues 1.
    Peter is in the second house. 2. Arnold is directly left of
   the one who only drinks water. 3. The one who only drinks
   water is directly left of the person who likes milk. We want
    to know who lives in each house and who drinks what.
The solution would be: {
"Peter": 2,
"Eric": 3,
"Arnold": 1,
"tea": 1,
"water": 2,
"milk": 3
```

```
}

Multishot - Convert:

... (Same as Baseline Convert)...

For this question: There are two houses. One is red, the other
    is blue. Person A lives in the red house, and Person B lives
    in the blue house. Who lives in each house?
{
"houses_count": 2,
"categories": {
"colors": ["Red", "Blue"],
"people": ["A", "B"]
},
"constraints": [
{
"type": "distinct_categories",
"categories": ["colors", "people"]
},
{
"type": "range",
"from": 1,
"to": 2
},
{"type": "eq", "var1": "PersonA", "var2": "Red"},
{"type": "eq", "var1": "PersonB", "var2": "Blue"}
]
}

For this question: here are 3 houses, numbered 1 to 3 from left
    to right. Each house is occupied by a different person. Each
    house has a unique attribute for each characteristic: - Names
    : Peter, Eric, Arnold - Drinks: tea, water, milk ## Clues 1.
     Peter is in the second house. 2. Arnold is directly left of
    the one who only drinks water. 3. The one who only drinks
    water is directly left of the person who likes milk. We want
```

```
    to know who lives in each house and who drinks what.
The solution would be: {
"houses_count": 3,
"categories": {
"names": ["Peter", "Eric", "Arnold"],
"drinks": ["tea", "water", "milk"]
},
"constraints": [
{
"type": "distinct_categories",
"categories": ["names", "drinks"]
},
{
"type": "range",
"from": 1,
"to": 3
},
{ "type": "eq", "var1": "Peter", "var2int": 2 },
{
"type": "eq_offset",
"var1": "water",
"var2": "Arnold",
"offset": 1
},
{
"type": "eq_offset",
"var1": "milk",
"var2": "water",
"offset": 1
}
]
}

Baseline Solve:

Below are the rules for solving:
```

```
Solve this puzzle by returning a Python/JSON dictionary that
    maps each distinct item (like colors or names) to an integer
    house index, where House #1 is the leftmost and House #N is
    the rightmost. For example, if there are two items: Red, Blue
     for houses left to right, and we know Red=1 (left), Blue=2 (
    right). If the puzzle also says PersonA=1, PersonB=2, the
    final dictionary is exactly:
{'Red':1, 'Blue':2, 'PersonA':1, 'PersonB':2}

Make sure there is No explanations, extra text or commentary in
    the solution key and just the dictionary in the EXACT format
    shown.

Baseline - Convert:

You are given a logic puzzle. Convert this puzzle into a JSON
    structure usable by a Z3 solver:
- The JSON must have: "houses_count", "categories", and "
    constraints".
- "categories" is a dictionary mapping each category name to a
    list of items (strings).
- "constraints" is a list of objects describing each constraint.
     Each constraint must use exactly one of the following types:
- "eq"
- "eq_offset"
- "neighbor"
- "neq"
- "ImmediateLeft"
- "ImmediateRight"
- "distinct_categories"
- "range"
- "rightOf"     (means var1 is to the right of var2)
- "leftOf"      (means var1 is to the left of var2)
- "abs_diff"    (means the absolute difference between var1 and
    var2 equals a given number)
- Use only the following keys for each constraint as needed: "
    type", "var1", "var2", "offset", "var2int", "diff", and "
```

```
    categories".
- For "distinct_categories", ALWAYS use the format:
{{
"type":"distinct_categories",
"categories":["colors","names",...]
}}
Do not include "var1" or "var2" in this constraint.
-Include exactly one "range" constraint with keys "from" and "to
    ".
For instance, if the puzzle has 5 houses, the range constraint
    is:
{
"type": "range",
"from": 1,
"to": 5
}
This ensures each item is an integer from 1..5.
- Do NOT invent new constraint types or items that aren't in the
     puzzle's categories.
- Do NOT include explanations, reasoning, or extraneous text.
    Only output valid JSON.
- If referencing a numeric house index for a 'neq' or 'eq'
    constraint, use 'var2int'. For example:
{
"type": "neq",
"var1": "X",
"var2int": 2
}
means X cannot equal 2.

Example output (dummy puzzle, not your real puzzle):
{{
"houses_count": 5,
"categories": {{
"colors": ["Red","Green","Blue","Yellow","White"],
"people": ["Alice","Bob","Charlie","Diana","Evan"]
}},
```

```
"constraints": [
{{
"type":"distinct_categories",
"categories":["colors","people"]
}},
{{
"type":"range",
"from":1,
"to":5
}},
{{ "type":"eq","var1":"Alice","var2":"Red" }},
{{ "type":"eq_offset","var1":"Blue","var2":"Green","offset":1
   }},
{{ "type":"neighbor","var1":"Bob","var2":"Alice" }}
]
}}

ONLY output valid JSON. No commentary.
Puzzle:
```

A.4 Writing and Editing Tools

To ensure clarity, fluency, and grammatical accuracy in the final document, the following tools were used during the editing process:

- **Grammarly:** Employed for basic spell checking, grammar correction, and stylistic refinement throughout the writing process.
- **ChatGPT-4o:** Used selectively to revise sections of the dissertation for improved academic tone, structural coherence, and formal expression. The following prompt was used during these edits:

  "Proofread this and rewrite it to follow a formal academic style."

A.5 Log Examples

This section presents two representative log entries captured during evaluation. Each log includes metadata, LLM responses, ground truth comparison, and task-level accuracy results. The first example comes from a simple 2x2 puzzle and shows a fully successful run; the second comes from a complex 6x6 puzzle where the model partially succeeded in solving but failed in constraint-based validation.

Log Entry Puzzle_2 (2x2, Fully Correct)


```
{
        "timestamp": "2025-04-12 23:16:51",
        "llm_provider": "deepseek",
        "puzzle": "puzzle_2",
        "puzzle_size": "2x2",
        "variant": "full_test",
        "strategy": "cot",
        "chain_of_thought": "Solve: We are given two houses...,
        "puzzle_ground_truth_dict": {
            "Red": 1,
            "Blue": 2,
            "PersonA": 1,
            "PersonB": 2
        },
        "solve_dict_str": "{\"Red\": 1, \"Blue\": 2, \"PersonA\": 1, \"PersonB\": 2}",
        "solve_response_time": 12.58,
        "solve_token_usage": 550,
        "solve_accuracy": 1.0,
        "solve_correct_fields": 4,
        "solve_total_fields": 4,
        "solve_parsed": {
            "Red": 1,
            "Blue": 2,
            "PersonA": 1,
            "PersonB": 2
        },
        "convert_constraints": "{\"houses_count\": 2, ...,
        "convert_response_time": 15.14,
        "convert_token_usage": 1315,
        "convert_solver_str": "{\"Red\": 1, \"Blue\": 2, ..",
        "convert_solver_accuracy": 1.0,
        "convert_correct_fields": 4,
        "convert_total_fields": 4,
        "convert_solver_parsed": {
            "Red": 1,
            "Blue": 2,
            "PersonA": 1,
            "PersonB": 2
        },
        "official_z3_constraints_present": true,
        "constraints_accuracy": 1.0,
        "constraints_correct_fields": 8,
        "constraints_total_fields": 8,
        "error": "N/A"
    },
```

Log Entry Puzzle_8 (6x6, Partial Failure)


```
{
        "timestamp": "2025-04-12 23:55:26",
        "llm_provider": "deepseek",
        "puzzle": "puzzle_8",
        "puzzle_size": "6x6",
        "variant": "full_test",
        "strategy": "cot",
        "chain_of_thought": "Solve: Starting with clue 23, the ...,
        "prompt": "There are 6 houses, numbered 1 to 6 from ...,
        "puzzle_ground_truth_dict": {
            "Carol": 3,
            "Eric": 5,
            "Peter": 2,
            "Bob": 4,
            "Arnold": 6,
            "classical": 2,
           ...
        },
        "solve_dict_str": "{\"Alice\": 1, \"Carol\": 3, ...,
        "solve_response_time": 34.0,
        "solve_token_usage": 1952,
        "solve_accuracy": 0.3611111111111111,
        "solve_correct_fields": 13,
        "solve_total_fields": 36,
        "solve_parsed": {
            "Alice": 1,
            "Carol": 3,
            "Eric": 5,
            "Peter": 4,
            "Bob": 6,
            ...
        },
        "convert_constraints": "{\"houses_count\": 6, \"categories...",
        "convert_response_time": 60.94,
        "convert_token_usage": 3307,
        "convert_solver_str": "N/A",
        "convert_solver_accuracy": 0.0,
        "convert_correct_fields": 0,
        "convert_total_fields": 36,
        "convert_solver_parsed": "N/A",
        "official_z3_constraints_present": true,
        "constraints_accuracy": 0.5789473684210527,
        "constraints_correct_fields": 22,
        "constraints_total_fields": 38,
        "error": "Z3 solver returned no solution for LLM constraints."
    },
```


A.6 Benchmark Aggregates Snapshot

The following snippet provides a summary of evaluation metrics aggregated across all runs. It includes both task-level and difficulty-level statistics for solve, convert, and constraint evaluation tasks. Metrics shown include count of samples, full accuracy, weighted (cell-level) accuracy, token usage, and difficulty-based breakdowns.

Benchmark Results Summary (JSON Format)


```
{
"solve": {
    "count": 50,
    "accuracy_sum": 41.0,
    "weighted_sum": 44.76888888888889,
    "total_fields": 750.0,
    "averages": {
        "average_accuracy": 0.82,
        "weighted_accuracy": 0.8953777777777777,
        "total_fields": 750.0,
        "entries": 50
    }
},
"convert": {
    "count": 50,
    "accuracy_sum": 43.0,
    "weighted_sum": 45.74166666666666,
    "total_fields": 750.0,
    "averages": {
        "average_accuracy": 0.86,
        "weighted_accuracy": 0.9148333333333332,
        "total_fields": 750.0,
        "entries": 50
    }
},
"constraints": {
    "count": 50,
    "accuracy_sum": 1.0,
    "weighted_sum": 7.768445443445444,
    "total_fields": 206.0,
    "averages": {
        "average_accuracy": 0.02,
        "weighted_accuracy": 0.15536890886890886,
        "total_fields": 206.0,
        "entries": 50
    }
},
"total_solve_tokens": 211786.0,
"total_convert_tokens": 250442.0,
"total_all_tokens": 462228.0,
"by_difficulty": {
    "X-Large": {
        "solve": {
            "count": 9,
            "accuracy_sum": 1.0,
            "weighted_sum": 4.393888888888888,
            "total_fields": 278.0,
            "averages": {
                "average_accuracy": 0.1111111111111111,
```

```
                "weighted_accuracy": 0.4882098765432098,
                "total_fields": 278.0,
                "entries": 9
            }
        },
        "convert": {
            "count": 9,
            "accuracy_sum": 5.0,
            "weighted_sum": 6.566666666666667,
            "total_fields": 278.0,
            "averages": {
                "average_accuracy": 0.5555555555555556,
                "weighted_accuracy": 0.7296296296296297,
                "total_fields": 278.0,
                "entries": 9
            }
        },
        "constraints": {
            "count": 9,
            "accuracy_sum": 1.0,
            "weighted_sum": 1.5769230769230769,
            "total_fields": 71.0,
            "averages": {
                "average_accuracy": 0.1111111111111111,
                "weighted_accuracy": 0.1752136752136752,
                "total_fields": 71.0,
                "entries": 9
            }
        }
    },
    "Small": {
        "solve": {
            "count": 19,
            "accuracy_sum": 19.0,
            "weighted_sum": 19.0,
            "total_fields": 132.0,
            "averages": {
                "average_accuracy": 1.0,
                "weighted_accuracy": 1.0,
                "total_fields": 132.0,
                "entries": 19
            }
        },
        "convert": {
            "count": 19,
            "accuracy_sum": 18.0,
            "weighted_sum": 18.0,
            "total_fields": 132.0,
            "averages": {
                "average_accuracy": 0.9473684210526315,
                "weighted_accuracy": 0.9473684210526315,
                "total_fields": 132.0,
                "entries": 19
            }
        },
```

```
        "constraints": {
            "count": 19,
            "accuracy_sum": 0.0,
            "weighted_sum": 4.039141414141414,
            "total_fields": 61.0,
            "averages": {
                "average_accuracy": 0.0,
                "weighted_accuracy": 0.21258639021796918,
                "total_fields": 61.0,
                "entries": 19
            }
        }
    },
    "Medium": {
        "solve": {
            "count": 15,
            "accuracy_sum": 14.0,
            "weighted_sum": 14.375,
            "total_fields": 215.0,
            "averages": {
                "average_accuracy": 0.9333333333333333,
                "weighted_accuracy": 0.9583333333333334,
                "total_fields": 215.0,
                "entries": 15
            }
        },
        "convert": {
            "count": 15,
            "accuracy_sum": 13.0,
            "weighted_sum": 14.175,
            "total_fields": 215.0,
            "averages": {
                "average_accuracy": 0.8666666666666667,
                "weighted_accuracy": 0.9450000000000001,
                "total_fields": 215.0,
                "entries": 15
            }
        },
        "constraints": {
            "count": 15,
            "accuracy_sum": 0.0,
            "weighted_sum": 2.1523809523809523,
            "total_fields": 67.0,
            "averages": {
                "average_accuracy": 0.0,
                "weighted_accuracy": 0.14349206349206348,
                "total_fields": 67.0,
                "entries": 15
            }
        }
    },
    "Large": {
        "solve": {
            "count": 7,
            "accuracy_sum": 7.0,
```

```
            "weighted_sum": 7.0,
            "total_fields": 125.0,
            "averages": {
                "average_accuracy": 1.0,
                "weighted_accuracy": 1.0,
                "total_fields": 125.0,
                "entries": 7
            }
        },
        "convert": {
            "count": 7,
            "accuracy_sum": 7.0,
            "weighted_sum": 7.0,
            "total_fields": 125.0,
            "averages": {
                "average_accuracy": 1.0,
                "weighted_accuracy": 1.0,
                "total_fields": 125.0,
                "entries": 7
            }
        },
        "constraints": {
            "count": 7,
            "accuracy_sum": 0.0,
            "weighted_sum": 0.0,
            "total_fields": 7.0,
            "averages": {
                "average_accuracy": 0.0,
                "weighted_accuracy": 0.0,
                "total_fields": 7.0,
                "entries": 7
            }
        }
    }
},
"time_range": {
    "earliest": "2025-04-14 19:55:00",
    "latest": "2025-04-15 00:30:33"
}
}
```

Example: Chain-of-Thought Reasoning Trace

The following entry illustrates a reasoning trace generated by Deepseek-Chat under the Chain-of-Thought (CoT) prompting strategy. This includes both the step-by-step explanation produced during solving, and the intermediate logic used in converting natural language clues into Z3-compatible constraints.

Chain-of-Thought Output (Solve + Convert)

Prompt: There are 2 houses, numbered 1 to 2 from left to right, as seen from across the street. Each house is occupied by a different person. Each house has a unique attribute for each of the following characteristics:- Each person has a unique name: ‘Eric‘, ‘Arnold‘- Everyone has something unique for lunch: ‘pizza‘, ‘grilled cheese‘ - People own unique car models: ‘tesla model 3‘, ‘ford f150‘ Clues: 1. Eric is in the first house. 2. The person who loves eating grilled cheese is not in the second house. 3. The person who owns a Ford F-150 is the person who is a pizza lover. Solve the puzzle: Who is in each house and what are their lunch and car model assignments?

Chain-of-Thought Reasoning: Solve: Starting with clue 1, we know Eric is in house 1, so Arnold must be in house 2. From clue 2, the grilled cheese lover is not in house 2, so they must be in house 1. Therefore, house 1 has grilled cheese, and house 2 must have pizza. From clue 3, the pizza lover owns the ford f150. Since house 2 has pizza, they own the ford f150. That leaves house 1 to own the tesla model 3.;

Convert: To solve this puzzle, let’s analyze the clues step by step. We have two houses, House 1 and House 2, each with distinct attributes across three categories: names, lunches, and car models. From clue 1, we know Eric is in House 1, so Arnold must be in House 2. From clue 2, the grilled cheese lover is not in House 2, so they must be in House 1, meaning House 2 has pizza for lunch. From clue 3, the Ford F-150 owner loves pizza, so House 2 has the Ford F-150, and House 1 must have the Tesla Model 3. This gives us the complete assignments: House 1 has Eric, grilled cheese, and Tesla Model 3; House 2 has Arnold, pizza, and Ford F-150.

Ground Truth: ”Eric”: 1, ”Arnold”: 2, ”grilled cheese”: 1, ”pizza”: 2, ”tesla model 3”: 1, ”ford f150”: 2